INTRODUÇÃO À COMPUTAÇÃO

UD II - ALGORITMOS

SUBROTINAS

Subrotinas

ELEMENTOS DE COMPETÊNCIA

- Empregar recursos para operar em ambientes humanizados, integrando as dimensões física, humana e informacional deste ambiente operacional.
- Tomar decisões e conduzir ações, em situações de crise.

UD II - INTRODUÇÃO À ALGORITMOS

Subrotinas

OBJETIVOS

1. Descrever funções e procedimentos. (CONCEITUAL)
2. Compreender a aplicação dos conceitos de funções e procedimentos na elaboração de algoritmos (modularização). (CONCEITUAL)
3. Aplicar os conceitos de funções e procedimentos na elaboração de algoritmos (modularização). (PROCEDIMENTAL)

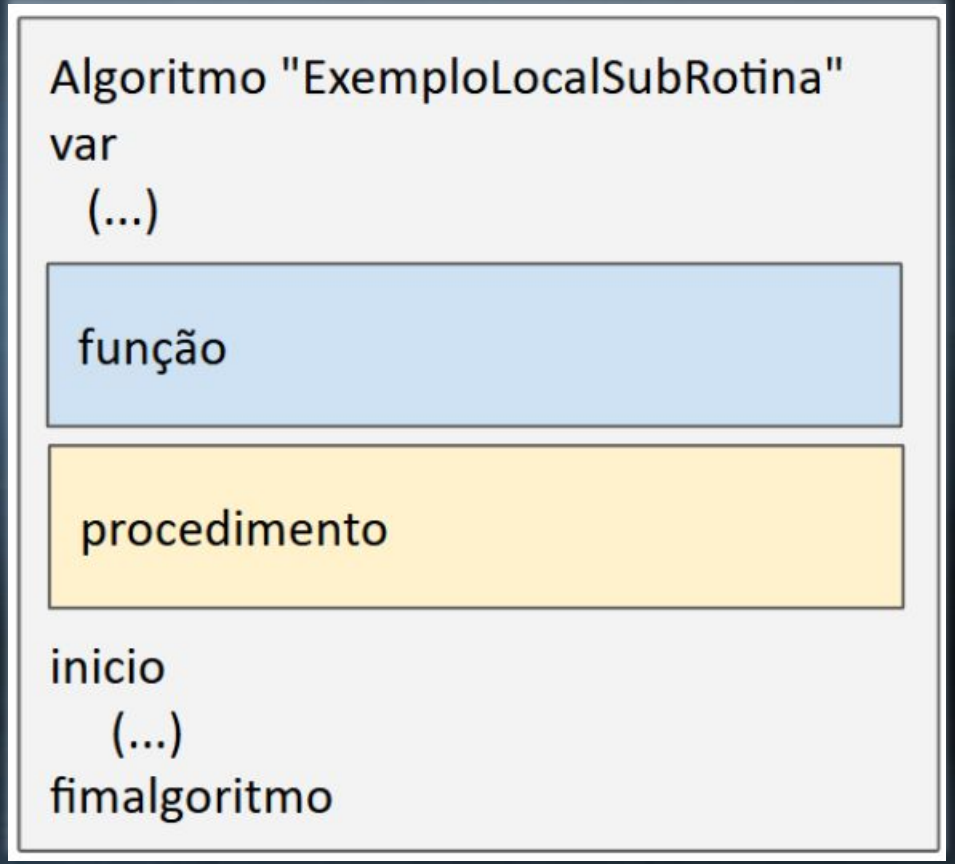
Sub-rotinas

ATITUDES

1. Organização: capacidade de desenvolver atividades de forma sistemática e eficiente.
2. Dedicação: agir, realizando espontaneamente, com empenho e entusiasmo, as atividades necessárias ao cumprimento da missão.
3. Responsabilidade: capacidade de cumprir suas atribuições assumindo e enfrentando as consequências de suas atitudes e decisões.

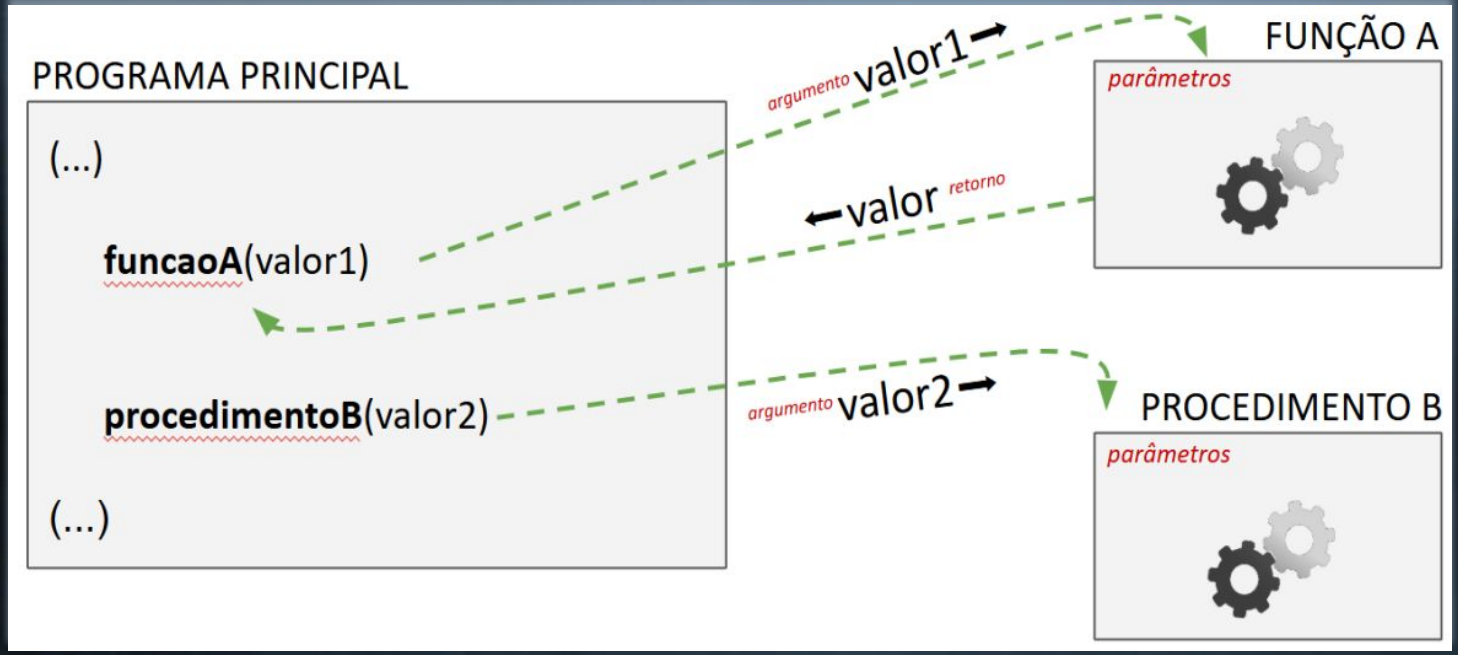
Sub-rotinas

Funções e procedimentos são sub-rotinas isoladas que executam tarefas específicas e podem ser utilizadas quando invocadas por um programa.



Sub-rotinas

Ao ser invocada (chamada) uma sub-rotina pelo programa principal, pode ser necessário informar argumentos para a sua execução, atendendo aos parâmetros da função.



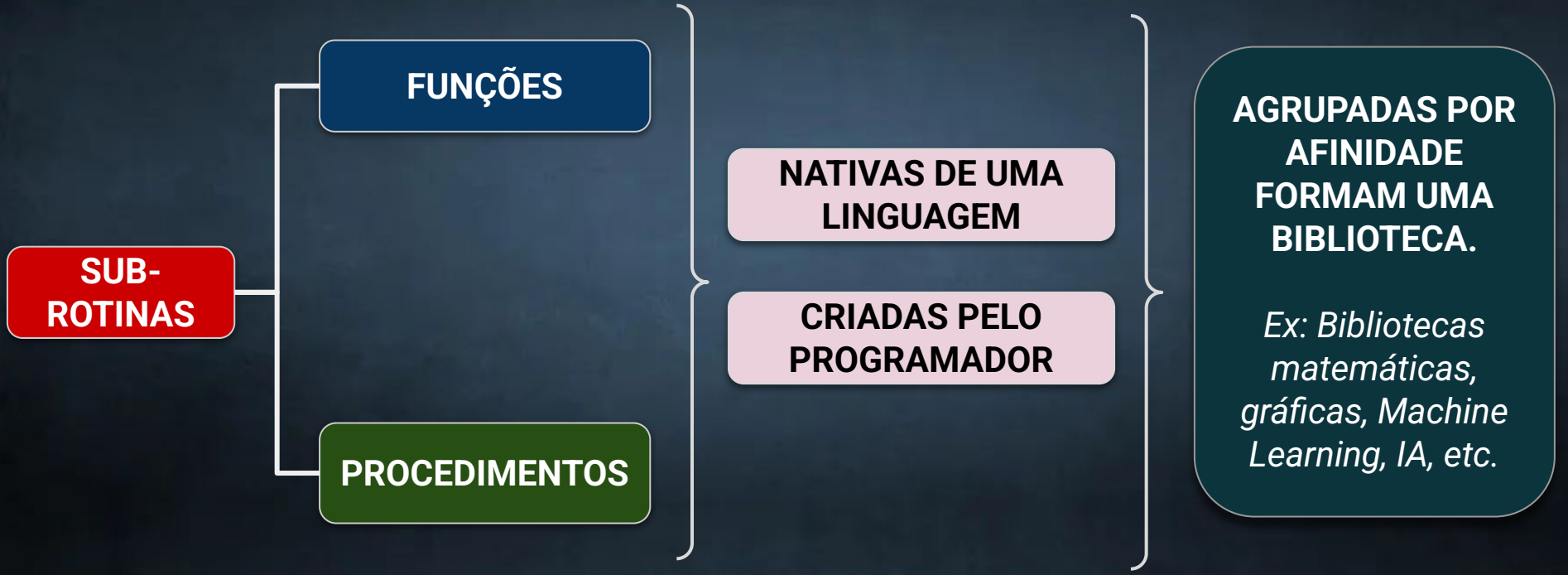
Vantagens da utilização de sub-rotinas

- Permite a reutilização do código, o que possibilita o aumento da produtividade, redução de erros e do tamanho dos programas e maior facilidade de manutenção.

- Redução da complexidade dos programas, onde tarefas complexas são decompostas em tarefas menores e com menor grau de complexidade.

- Aumento da legibilidade do programa, principalmente se o nome da sub-rotina oferece uma boa noção da tarefa que ela executa.

Taxonomia / sub-rotinas



Função

Uma função é uma unidade de código de programa autônoma desenvolvida para cumprir uma determinada tarefa em particular.

Funções podem ser referenciadas (chamadas ou invocadas) em qualquer parte do programa principal, por meio de seu nome, e que retornam um valor ao programa que a invocou. Ao término da execução da função, o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que efetuou a chamada da sub-rotina.

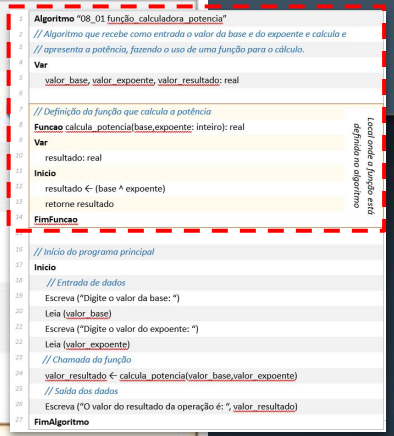
Função

Sintaxe de definição de uma função:

```
Função <nome>(<parâmetros>:<tipo de dados dos parâmetros>): <tipo de dados de retorno>
Var
        <variáveis locais da função>: <tipos das variáveis locais>
Inicio
        <corpo da função>
FimFuncao
```

```
1  Algoritmo "08_01 função_calculadora_potencia"
2  // Algoritmo que recebe como entrada o valor da base e do expoente e calcula
3  // apresenta a potência, fazendo o uso de uma função para o cálculo.
4  Var
5      valor_base, valor_expoente, valor_resultado: real
6
7  // Definição da função que calcula a potência
8  Funcao calcula_potencia(base,expoente: inteiro): real
9  Var
10     resultado: real
11 Inicio
12     resultado ← (base ^ expoente)
13     retorne resultado
14 FimFuncao
15
```

Local onde a função está definida no algoritmo



ALGORITMO NO VISUALG

ALGORITMO NO VISUALG

```
16  // Início do programa principal
17  Inicio
18      // Entrada de dados
19      Escreva ("Digite o valor da base: ")
20      Leia (valor_base)
21      Escreva ("Digite o valor do expoente: ")
22      Leia (valor_expoente)
23      // Chamada da função
24      valor_resultado ← calcula_potencia(valor_base,valor_expoente)
25      // Saída dos dados
26      Escreva ("O valor do resultado da operação é: ", valor_resultado)
27  FimAlgoritmo
```

```
Algoritmo "08_01 função_calculadora_potencia"
// Algoritmo que recebe como entrada o valor da base e do expoente e calcula e
// apresenta a potência, fazendo o uso de uma função para o cálculo.
Var
   valor_base, valor_expoente, valor_resultado: real

// Definição da função que calcula a potência
Funcao calcula_potencia(base,expoente: inteiro): real
Var
   resultado: real
Inicio
   resultado ← (base ^ expoente)
   retorne resultado
FimFuncao

// Início do programa principal
Inicio
   // Entrada de dados
   Escreva ("Digite o valor da base: ")
   Leia (valor_base)
   Escreva ("Digite o valor do expoente: ")
   Leia (valor_expoente)
  // Chamada da função
   valor_resultado ← calcula_potencia(valor_base,valor_expoente)
   // Saída dos dados
   Escreva ("O valor do resultado da operação é: ", valor_resultado)
FimAlgoritmo
```

Local onde a função está definida no algoritmo

Variáveis globais

Variável local

retorno de um valor

```
1  Algoritmo "08_01 função_calculadora_potencia"
2  // Algoritmo que recebe como entrada o valor da base e do expoente e calcula e
3  // apresenta a potência, fazendo o uso de uma função para o cálculo.
4  Var
5     valor_base, valor_expoente, valor_resultado: real
6
7  // Definição da função que calcula a potência
8  Funcao calcula_potencia(base,expoente: inteiro): real
9  Var
10    resultado: real
11 Inicio
12    resultado ← (base ^ expoente)
13    retorne resultado
14 FimFuncao
15
16 // Início do programa principal
17 Inicio
18    // Entrada de dados
19    Escreva ("Digite o valor da base: ")
20    Leia (valor_base)
21    Escreva ("Digite o valor do expoente: ")
22    Leia (valor_expoente)
23   // Chamada da função
24    valor_resultado ← calcula_potencia(valor_base,valor_expoente)
25    // Saída dos dados
26    Escreva ("O valor do resultado da operação é: ", valor_resultado)
27 FimAlgoritmo
```

Local onde a função está definida no algoritmo

função invocada com a passagem dos argumentos

Procedimentos

São sub-rotinas que executam instruções sem retorno de qualquer valor ao programa que o invocou.

É um bloco de programa identificado por um nome, por meio do qual será invocado em qualquer parte do programa principal. Quando chamada por um programa, ele é executado e ao seu término o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que efetuou a chamada da sub-rotina.

Procedimentos

Sintaxe de definição de um procedimento:

```
Procedimento <nome>(<parâmetros>:<tipo de dados dos parâmetros>): <tipo de dados de retorno>
Var
        <variáveis locais do procedimento>: <tipos das variáveis locais>
Inicio
        <corpo do procedimento>
FimFuncao
```

```
1  Algoritmo "08_02 Exemplo de Procedimento"
2  // Algoritmo que demonstra o uso de procedimento
3  // Exibe um cabeçalho padrão antes do início do programa
4  Var
5      nomeUsuario: caractere
6
7  // Definição do procedimento mostra cabeçalho padrão
8  Procedimento mostraMensagem(mensagem: caractere)
9  Inicio
10     escreval("==================================")
11     escreval("= ", mensagem)
12     escreval("==================================")
13 FimProcedimento
14
15 // Início do programa principal
16 Inicio
17     mostraMensagem("Programa teste v1.0")          // Chamada do procedimento
18
19     Escreva("Digite seu nome:")
20     Leia(nomeUsuario)
21     mostraMensagem ("Olá, " , nomeUsuario, "! Seja bem vindo ao nosso sistema!")
22     mostraMensagem("Fim do programa")              // Chamada do procedimento
23 FimAlgoritmo
```

Local onde o procedimento está definido no algoritmo

SEM retorno de um valor

procedimento invocado com passagem de um argumento

Fluxo de função e procedimento

```
Algoritmo "UtilizandoSubrotinas"
Var
        x, k: inteiro
Início
        escreva("Informe um valor para x:")
        leia(x)
        k ← elevaAoCubo(x)

        escreva("Valor elevado ao cubo: ", k)

        mensagemFinal("Fim de Programa!")
Fim
```

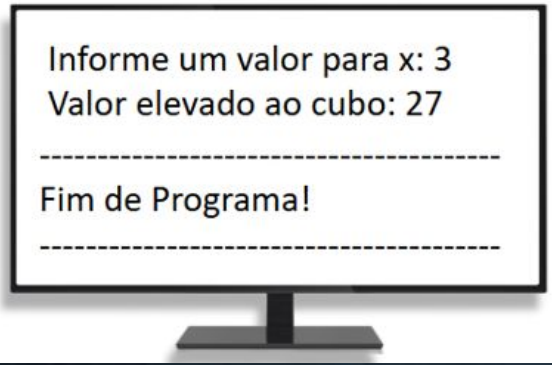
3

27

"Fim de Programa!"

```
Função elevaAoCubo(base:inteiro): inteiro
Var
        calculo: inteiro
Início
        calculo <- base ^ 3
        retorne calculo
Fim
```

```
Procedimento mensagemFinal(mensagem: texto)
Início
        escreva("----------------------------------------")
        escreva(mensagem)
        escreva("----------------------------------------")
Fim
```



Aplicação de Funções e Procedimentos

Uma maneira simples de realizar a modularização do programa, que consiste em dividir um problema grande e complexo em problemas menores e mais simples (sub-rotinas), facilitando seu desenvolvimento, entendimento e manutenção.

Os trechos de código menores e mais simples, quando integrados, possibilitam a resolução do problema como um todo. Portanto, a modularização se baseia no princípio de “dividir para conquistar”.

Aplicação de Funções e Procedimentos

Um exemplo de programa que gerencia uma folha de pagamento:

Obter os dados dos funcionários.
Calcular INSS.
Calcular FGTS.
Calcular imposto de renda.
Calcular férias.
Calcular salário líquido.
Imprimir os contracheques.

Aplicação de Funções e Procedimentos

- Redução da complexidade (maior simplicidade), já que os vários módulos tendem a ser mais simples.
- Melhor entendimento (maior legibilidade), pois a modularização torna o programa mais fácil de entender.
- Maior manutenibilidade (facilidade de manutenção).
- Aumento da possibilidade de reutilização, permitindo que uma determinada sub-rotina de um programa seja reaproveitada.
- Maior facilidade de desenvolvimento, pois é mais fácil pensar na solução de problemas menores e mais simples.

Bibliotecas de sub-rotinas

- São conjuntos de recursos pré-codificados, organizados em módulos, que podem ser integrados a novos projetos. O principal objetivo é a produtividade: ao invés de reescrever códigos do zero, o programador utiliza soluções já testadas e otimizadas.
- Categorizadas por domínio de aplicação: cálculos matemáticos, processamento estatístico, renderização gráfica, automação, Machine Learning e Inteligência Artificial.
- Quanto à origem, elas se dividem em duas categorias principais:
  - Bibliotecas Nativas: Conjunto de funções internas que já vêm instaladas com a linguagem de programação.
  - Bibliotecas de Terceiros: Pacotes desenvolvidos por empresas ou pela comunidade open source.

Exercícios em sala 1

Lista 8.1

Utilizando os conceitos de procedimentos e funções, desenvolva um algoritmo que calcule, a partir de 2 números inteiros positivos P e M, a combinação de M elementos tomados P a P, conforme a expressão:

$$C = \frac{M!}{P!(M-P)!}$$

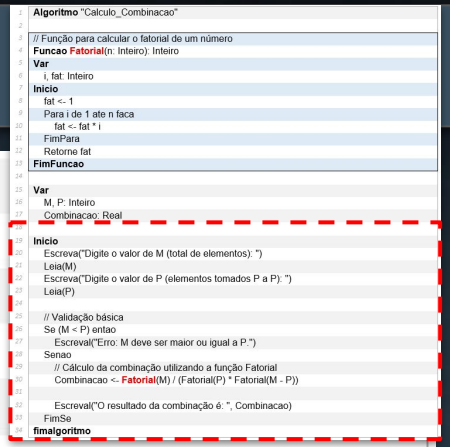
Os valores dos diversos fatoriais existentes na expressão deverão ser obtidos por meio de uma função para isso destinada.

Exercícios em sala 1 - Solução

```
1  Algoritmo "Calculo_Combinacao"
2
3  // Função para calcular o fatorial de um número
4  Funcao Fatorial(n: Inteiro): Inteiro
5  Var
6     i, fat: Inteiro
7  Inicio
8     fat <- 1
9     Para i de 1 ate n faca
10       fat <- fat * i
11    FimPara
12    Retorne fat
13 FimFuncao
14
15 Var
16    M, P: Inteiro
17    Combinacao: Real
```

Exercícios em sala 1 - Solução

```
19  Inicio
20      Escreva("Digite o valor de M (total de elementos): ")
21      Leia(M)
22      Escreva("Digite o valor de P (elementos tomados P a P): ")
23      Leia(P)
24
25      // Validação básica
26      Se (M < P) entao
27          Escreval("Erro: M deve ser maior ou igual a P.")
28      Senao
29          // Cálculo da combinação utilizando a função Fatorial
30          Combinacao <- Fatorial(M) / (Fatorial(P) * Fatorial(M - P))
31
32          Escreval("O resultado da combinação é: ", Combinacao)
33      FimSe
34  fimalgoritmo
```



```
Algoritmo "Calculo_Combinacao"

// Função para calcular o fatorial de um número
Funcao Fatorial(n: Inteiro): Inteiro
Var
   i, fat: Inteiro
Inicio
   fat <- 1
   Para i de 1 ate n faca
      fat <- fat * i
   FimPara
   Retorne fat
FimFuncao

Var
   M, P: Inteiro
   Combinacao: Real

Inicio
   Escreva("Digite o valor de M (total de elementos): ")
   Leia(M)
   Escreva("Digite o valor de P (elementos tomados P a P): ")
   Leia(P)

   // Validação básica
   Se (M < P) entao
      Escreval("Erro: M deve ser maior ou igual a P.")
   Senao
      // Cálculo da combinação utilizando a função Fatorial
      Combinacao <- Fatorial(M) / (Fatorial(P) * Fatorial(M - P))

      Escreval("O resultado da combinação é: ", Combinacao)
   FimSe
fimalgoritmo
```

Exercícios em sala 2

Implemente um algoritmo que calcula as raízes de uma equação do segundo grau do tipo ax^2+bx+c, utilizando a fórmula de Bhaskara. O algoritmo deverá receber via teclado os valores dos índices a, b e c da equação e mostrar mensagem informando a quantidade de raízes (caso possua), com os seus respectivos valores ou que a equação não possui raízes reais.

Dado: fórmula de Bhaskara: $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ *. Exemplos de saída:*

Lista 8.1 - Modifique o algoritmo do exercício 6.11 da lista de exercícios referente ao capítulo 6 (Condicionais) da apostila de Algoritmos, de modo que o cálculo do valor de delta, o cálculo do valor das raízes (quando houverem) e a impressão dos resultados, sejam realizadas por meio de sub-rotinas, que serão chamadas pelo programa principal.

```
Algoritmo "Equacao_Segundo_Grau_Subrotinas"

Var
   a, b, c, valorDelta: Real

// 1. Função para calcular o valor de Delta
Funcao CalcularDelta(a, b, c: Real): Real
Inicio
   Retorne (b * b) - (4 * a * c)
FimFuncao

// 2. Procedimento para calcular e mostrar as raízes
Procedimento CalcularRaizes(a, b, delta: Real)
Var
   x1, x2: Real
Inicio
   Se (delta < 0) entao
      Escreval("A equação não possui raízes reais.")
   Senao
      Se (delta = 0) entao
         x1 <- (-b) / (2 * a)
         Escreval("A equação possui uma raiz real: ", x1)
      Senao
         x1 <- (-b + RaizQ(delta)) / (2 * a)
         x2 <- (-b - RaizQ(delta)) / (2 * a)
         Escreval("A equação possui duas raízes: ", x1, " e ", x2)
      FimSe
   FimSe
FimProcedimento

// --- Programa Principal ---
Inicio
   Escreva("Informe o valor de a: ")
   Leia(a)

   // O valor de 'a' não pode ser zero em uma equação do segundo grau
   Se (a = 0) entao
      Escreval("O valor de 'a' deve ser diferente de zero.")
   Senao
      Escreva("Informe o valor de b: ")
      Leia(b)
      Escreva("Informe o valor de c: ")
      Leia(c)

      // Chamada da função para obter o Delta
      valorDelta <- CalcularDelta(a, b, c)

      // Chamada do procedimento para processar o resultado
      Escreval("-----------------------------------------")
      CalcularRaizes(a, b, valorDelta)
   FimSe
fimalgoritmo
```

Função

Procedimento

Função CalcularDelta

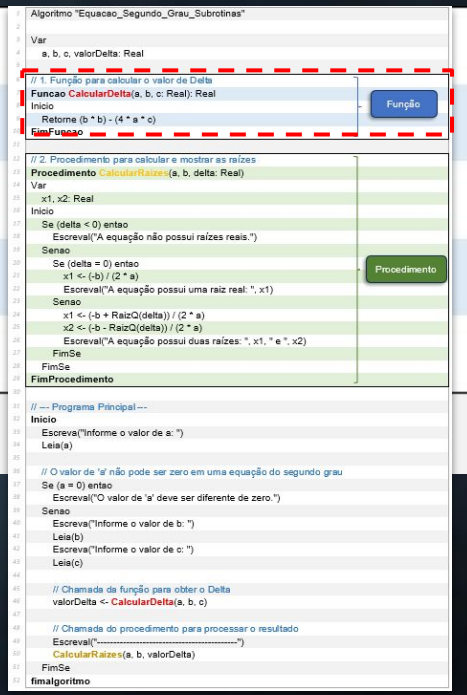
Procedimento CalcularRaízes

Programa Principal

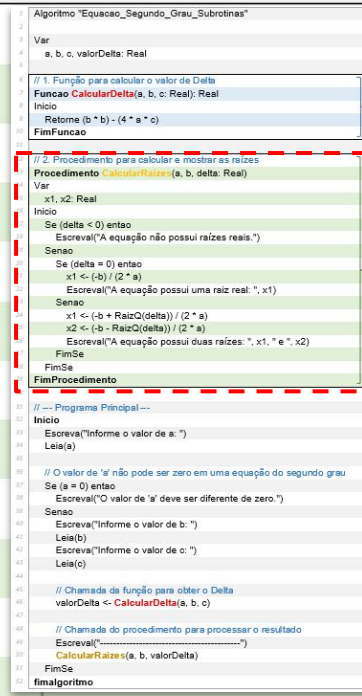
Solução Exercício 2

Sol. Ex. 2

```
6   // 1. Função para calcular o valor de Delta
7   Funcao CalcularDelta(a, b, c: Real): Real
8   Inicio
9       Retorne (b * b) - (4 * a * c)
10  FimFuncao
11
```

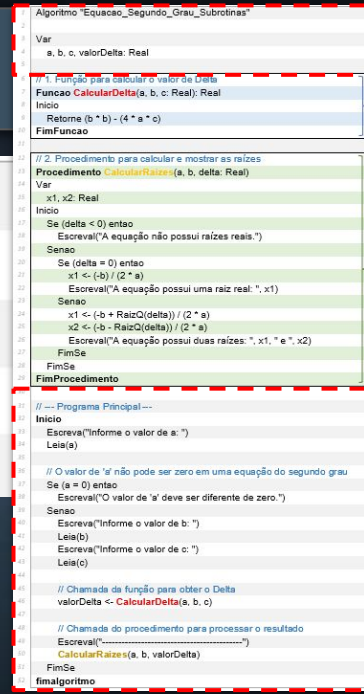


```
// 2. Procedimento para calcular e mostrar as raízes
Procedimento CalcularRaizes(a, b, delta: Real)
Var
   x1, x2: Real
Inicio
   Se (delta < 0) entao
      Escreval("A equação não possui raízes reais.")
   Senao
      Se (delta = 0) entao
         x1 <- (-b) / (2 * a)
         Escreval("A equação possui uma raiz real: ", x1)
      Senao
         x1 <- (-b + RaizQ(delta)) / (2 * a)
         x2 <- (-b - RaizQ(delta)) / (2 * a)
         Escreval("A equação possui duas raízes: ", x1, " e ", x2)
      FimSe
   FimSe
FimProcedimento
```



Sol. Ex. 2

```
1  Algoritmo "Equacao_Segundo_Grau_Subrotinas"
2
3  Var
4     a, b, c, valorDelta: Real
5
```



```
Algoritmo "Equacao_Segundo_Grau_Subrotinas"

Var
   a, b, c, valorDelta: Real

// 1. Função para calcular o valor de Delta
Funcao CalcularDelta(a, b, c: Real): Real
Inicio
   Retorne (b * b) - (4 * a * c)
FimFuncao

// 2. Procedimento para calcular e mostrar as raízes
Procedimento CalcularRaizes(a, b, delta: Real)
Var
   x1, x2: Real
Inicio
   Se (delta < 0) entao
      Escreval("A equação não possui raízes reais.")
   Senao
      Se (delta = 0) entao
         x1 <- (-b) / (2 * a)
         Escreval("A equação possui uma raiz real: ", x1)
      Senao
         x1 <- (-b + RaizQ(delta)) / (2 * a)
         x2 <- (-b - RaizQ(delta)) / (2 * a)
         Escreval("A equação possui duas raízes: ", x1, " e ", x2)
      FimSe
   FimSe
FimProcedimento

// --- Programa Principal ---
Inicio
   Escreva("Informe o valor de a: ")
   Leia(a)

   // O valor de 'a' não pode ser zero em uma equação do segundo grau
   Se (a = 0) entao
      Escreval("O valor de 'a' deve ser diferente de zero.")
   Senao
      Escreva("Informe o valor de b: ")
      Leia(b)
      Escreva("Informe o valor de c: ")
      Leia(c)

      // Chamada da função para obter o Delta
      valorDelta <- CalcularDelta(a, b, c)

      // Chamada do procedimento para processar o resultado
      Escreval("---------------------------------------")
      CalcularRaizes(a, b, valorDelta)
   FimSe
fimalgoritmo
```

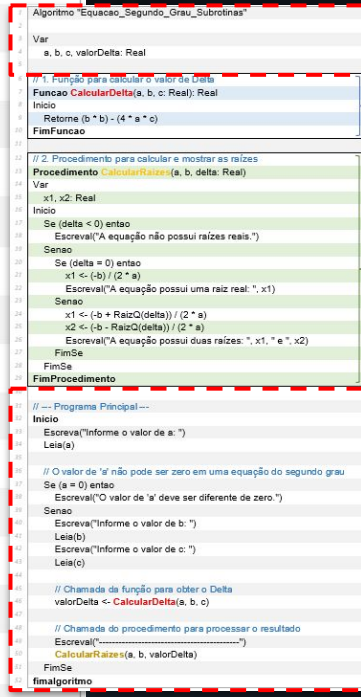
Sol. Ex. 2

```
// --- Programa Principal ---
Inicio
   Escreva("Informe o valor de a: ")
   Leia(a)

   // O valor de 'a' não pode ser zero em uma equação do segundo grau
   Se (a = 0) entao
      Escreval("O valor de 'a' deve ser diferente de zero.")
   Senao
      Escreva("Informe o valor de b: ")
      Leia(b)
      Escreva("Informe o valor de c: ")
      Leia(c)

      // Chamada da função para obter o Delta
      valorDelta <- CalcularDelta(a, b, c)

      // Chamada do procedimento para processar o resultado
      Escreval("----------------------------------------")
      CalcularRaizes(a, b, valorDelta)
   FimSe
fimalgoritmo
```



Sol. Ex. 2