5. Funções

5.1 Introdução

O exercício 7 da seção 3.5 solicitava que fosse feito um programa para a determinação das raízes reais ou complexas de uma equação do segundo grau. Com os conhecimentos adquiridos até a referida seção, um possível programa seria o seguinte.

```
//Programa para a determinação das raízes de uma equação do segundo grau
Variaveis
    Numerico a, b, c, x1, x2, Delta, Real, Imag
Inicio
    Escrever "Digite os coeficientes"
    Ler a
    Ler b
    Ler c
    Se a <> 0 entao
        Real = -b/(2*a)
        Delta = b^2 - 4*a*c
        Se Delta >= 0 entao
            Imag = Raiz(Delta, 2)/(2*a)
            x1 = Real + Imag
            x2 = real - Imag
            Escrever "Raízes: ", x1, " e ", x2
        Senao
            Imag = Raiz(-Delta, 2)/(2*a)
            Escrever "Raízes: ", Real, " + ",Imag,"i e ", Real, " - ",Imag,"i"
        Fim_se
    Senao
        Escrever "A equação não e do segundo grau"
    Fim_se
Fim
```

Observe que os comandos Imag = Raiz(Delta, 2)/(2*a) e Imag = Raiz(-Delta, 2)/(2*a) são basicamente idênticos e são necessários para que o radicando seja positivo. O sistema ILA permite que uma ação deste tipo possa ser realizada "à parte do programa" e que este "chame" a execução desta ação quando necessário. Estas ações que são executadas "fora" do programa propriamente dito são realizadas através de *funções* que, como dito na seção 2.5, devem ser definidas logo após às definições das variáveis com a seguinte sintaxe:

```
Funcao Identificador(Lista de parâmetros)
Inicio
    //Sequência de comandos
Fim
```

Nesta definição *Lista de parâmetros* é um conjunto de variáveis (e, portanto, devem ser declaradas na área para tal) que receberão os valores para os quais a execução da função é solicitada, atuando como se fosse a "entrada" da função. Estes valores estes que são chamados *argumentos* da execução da função Como uma entrada de um programa, a lista de parâmetros de uma função pode ser vazia.

A execução de uma função pode ser solicitada com a colocação do identificador da função (e a lista dos *argumentos*) dentro de uma expressão ou numa linha do programa, como se faz com um comando. Para que uma função possa ser *ativada* (ou seja, tenha a sua execução solicitada) dentro de uma expressão é necessário que um dos seus comandos seja um comando cuja sintaxe é

Retornar Expressão

sendo o valor de *Expressão* utilizado para a avaliação da expressão que contém a *chamada* da função.

5.2 Exemplos Parte IV

1. Com o uso de funções, o programa da equação do segundo grau poderia ser melhorado com a utilização de uma função que calculasse o valor absoluto de Delta, permitindo que fosse calculada "a parte imaginária" da raiz, quer Delta seja positivo ou negativo.

```
//Programa para determinação das raízes de uma equação do segundo grau.
Variaveis
    Numerico a, b, c, x1, x2, Delta, Real, Imag, z, x
    //Função que retorna o valor absoluto de um número
    Funcao Abs(x)
    Inicio
        Se x < 0 entao
            Retornar -x
        Senao
            Retornar x
        Fim_se
    Fim
Inicio //Programa principal
    Escrever "Digite os coeficientes"
    Ler a
    Ler b
    Ler c
    Se a <> 0 entao
        Real = -b/(2*a)
        Delta = b^2 - 4*a*c
        z = Abs(Delta)
        Imag = Raiz(z, 2)/(2*a)
        Se Delta >= 0 entao
            x1 = Real + Imag
            x2 = real - Imag
            Escrever "Raízes: ", x1, " e ", x2
        Senao
            Escrever "Raízes: ", Real, " + ",Imag,"i e ", Real, " - ",Imag,"i"
        Fim_se
    Senao
        Escrever "A equação não é do segundo grau"
    Fim_se
Fim
```

2. De um modo geral, os sistemas de computação não trabalham com números racionais na forma de fração ordinária. A manipulação de frações ordinárias é feita considerando-se separadamente os termos da fração. Um programa que pretendesse simplificar uma fração ordinária poderia ter uma função que retornasse o máximo divisor comum dos termos da fração e a simplificação poderia ser feita dividindo-se os termos da fração por este máximo divisor comum.

```
//Programa para simplificar frações ordinárias
Variaveis
    Numerico Num, Den, x, y, r, Mdc, NovoNum, NovoDen
    //Função que retorna o máximo divisor comum de dois números dados
```

```
        Funcao MaxDivCom(x, y)
        Inicio
                r = Resto(x, y)
                Faca enquanto r <> 0
                        x = y
                        y = r
                        r = Resto(x, y)
                Fim_enquanto
                Retornar y
        Fim
Inicio //Programa principal
        Escrever "Digite os termos da fração"
        Ler Num
        Ler Den
        Mdc = MaxDivCom(Num, Den)
        NovoNum = Num/Mdc
        NovoDen = Den/Mdc
        Escrever "A fração ", Num, "/", Den, " simplificada e: ", NovoNum, "/", NovoDen
Fim
```

3. Um programa para listar todos os números primos menores do que um inteiro dado poderia ter uma *função lógica* (ou seja, uma função que retorna um valor *falso* ou um valor *verdadeiro*) que recebendo como argumento um número inteiro verificasse se este inteiro é ou não primo.

```
//Programa para listar todos os primos menores que um inteiro positivo dado
Variaveis
        Numerico Num, x, i, j
        //Função lógica que verifica se um número inteiro é primo
        Funcao Primo(x)
        Inicio
                i = 2
                Faca enquanto (Resto(x, i) <> 0) e (i < x/2)
                        i = i + 1
                Fim_enquanto
                Se (Resto(x, i) = 0) e (x <> 2) entao
                        Retornar falso
                Senao
                        Retornar verdadeiro
                Fim_se
        Fim
Inicio //Programa principal
        Escrever "Digite o número"
        Ler Num
        Escrever "Os primos menores que ", Num," são: "
        Para j = 2 ate Num - 1
                Se Primo(j) = verdadeiro entao
                        Escrever j
                Fim_se
        Proximo
Fim
```

4. Se quisermos um programa que gere uma tabela de *fatoriais*, podemos escrever uma função que recebendo um inteiro positivo retorna o fatorial deste inteiro. Lembrando que n! = 1 . 2 . ... . n e que 0! = 1, a função deve inicializar uma variável f com o valor 1 e numa estrutura de repetição de

i = 1 até n calcular f = f*i. Ao fim da estrutura de repetição f armazenará o fatorial procurado.

```
//Programa que gera uma tabela de fatoriais
Variaveis
    Numerico i, n, j, f, Num
    //Funcao que retorna o fatorial de um inteiro dado
    Funcao Fat(n)
    Inicio
        f = 1
        Para i = 1 ate n
            f = f*i
        Proximo
        Retornar f
    Fim
Inicio //Programa principal
    Escrever "Digite o valor máximo da tabela"
    Ler Num
    Para j = 0 ate Num
        Escrever j,"! =  ", Fat(j)
    Proximo
Fim
```

5. O exemplo 7 da seção 2.11 apresentou um programa que fornecia a parte fracionária de um inteiro dado, programa este que utilizava a função pré-definida Inteiro. Uma função que retorna a parte fracionária de um número positivo, sem usar nenhuma função pré-definida, poderia encontrar o maior inteiro menor que o número dado e retornar a diferença entre este número dado e o maior inteiro determinado.

```
//Função que determina a parte fracionária de um número e não utiliza funções pré-definidas.
Variaveis
    Numerico Num, x, i
Funcao Frac(x)
Inicio
    i = 0
    Faca enquanto i <= x
        i = i + 1
    Fim_enquanto
        i = i - 1
    Retornar x - i
Fim
```

6. As funções dos exemplos acima retornam, através do comando *Retornar*, um valor e elas são ativadas dentro de uma expressão. Pode-se definir funções que executem alguma tarefa, mas não retorne nenhum valor específico. Neste caso, a função é ativada com a referência ao seu identificador seguido, se for o caso, da lista de argumentos. A construção de um *menu de opções* de um programa que realize diversas tarefas é um exemplo deste tipo de função. Por exemplo, um programa que gerencie o acervo de uma biblioteca poderia conter a seguinte função.

```
Funcao Menu()
Inicio
    Escrever "1-Cadastrar usuário"
    Escrever "2-Cadastrar livro"
    Escrever "3-Empréstimo"
    Escrever "4-Devolução"
    Escrever "5-Encerrar"
```

```
        Escrever “Digite sua opção”
        Ler Opcao
    Fim
```

Neste caso, um dos primeiros comandos do programa principal seria a ativação da função através de

```
Menu()
```

como se escreve um comando. Observe que esta função também exemplifica uma função cuja lista de parâmetros é vazia.

7. A maior vantagem do uso de funções é a possibilidade de que um programa seja escrito em *módulos*, o que facilita a *legibilidade* do programa, a *manutenção* do programa (no sentido de que alterações no sistema são realizadas simplesmente alterando-se algumas funções) e permite que vários programadores desenvolvam um mesmo programa, cada um deles ficando responsável por módulos específicos. Para exemplificar, suponhamos que pretendemos desenvolver um *software matemático* para a manipulação algébrica de *números complexos*. Em ILA um programa como este poderia ter uma função que exibisse um menu de opções e funções para cada uma das tarefas que se pretende que o programa seja capaz de executar. Algo como

```
//Programa para álgebra dos números complexos
Variaveis
    Numerico a, b, c, d, r, i, x, y, Mod, Ang
    Caracter Opc
    Funcao Menu()
    Inicio
        Escrever "1-Módulo de um complexo"
        Escrever "2-Soma de dois complexos"
        Escrever "3-Produto de dois complexos"
        Escrever "4-Forma polar de um complexo"
        Escrever  "5-Encerra o programa"
        Escrever  "Digite sua opção"
        Ler Opc
    Fim
    Funcao LerDoisComplexos()
    Inicio
        Escrever "Digite os complexos"
        Ler a
        Ler b
        Ler c
        Ler d
    Fim
    Funcao Modulo(r, i)
    Inicio
        Retornar Raiz(a^2 + b^2, 2)
    Fim
    Funcao SomaComplexos(a, b, c, d)
    Inicio
        r = a + c
        i = b + d
    Fim
    Funcao MultiplicaComplexos(a, b, c, d)
    Inicio
        r = a*c - b*d
        i = a*d + b*c
```

```
        Fim
        Funcao FormaPolar(a, b)
        Inicio
            Mod = Modulo(a, b)
            Se a <> 0 entao
                Ang = Atan(b/a)
                Escrever a," + ",b , "i = ", Mod,".(cos(",Ang, ") + i.sen(", Ang,")"
            Senao
                Escrever b , "i = ", Mod,".(cos(Pi/2) + i.sen(Pi/2)"
            Fim_se
        Fim
    //Programa principal
    Inicio
        Faca enquanto Opc <> "5"
        Menu()
        Faca caso
            Caso Opc = "1":
                Escrever "Digite o complexo"
                Ler a
                Ler b
                Escrever "|",a,"+",b,"i| = ", Modulo(a, b)
            Caso Opc = "2":
                LerDoisComplexos()
                SomaComplexos(a, b, c, d)
                Escrever "(", a, " + ",b, "i) + (", c, " + ", d, "i) = ", r, " + ",i, "i"
            Caso Opc = "3":
                LerDoisComplexos()
                MultiplicaComplexos(a, b, c, d)
                Escrever "(", a, " + ",b, "i) . (", c, " + ", d, "i) = ", r, " + ",i, "i"
            Caso Opc = "4":
                Escrever "Digite o complexo"
                Ler a
                Ler b
                FormaPolar(a, b)
            Fim_caso
        Fim_enquanto
    Fim
```

Observe que este exemplo mostra que uma função pode chamar a ativação de outra função definida anteriormente, como a função FormaPolar chamou a função Modulo.

5.3 Recursividade

O exercício 12 da seção 4.5 definiu a *sequência de Fibbonaci* como sendo a sequência (a_n) definida por

$$a_n = \begin{cases} 1, n = 1 \, ou \, n = 2 \\ a_{n-1} + a_{n-2}, se \, n > 2 \end{cases}$$

Observe que o termo de ordem n é definido a partir de termos anteriores. Isto significa que para o cálculo de um determinado termo há necessidade de que se *recorra* a valores de todos os termos anteriores. Por exemplo, para a determinação de a_5 necessitamos conhecer a_4 e a_3; para a determinação destes dois, necessitamos conhecer a_2 e a_1. Uma definição com estas características é

dita uma definição por *recorrência* ou uma definição *recursiva*.

Outro ente matemático que pode ser definido por recorrência é o *fatorial* de um número natural, objeto de discussão do exemplo 4 da seção anterior. Embora se possa definir fatorial de um natural n por n! = 1 . 2 . 3 . ... . n, é matematicamente mais elegante definir por

$$n! = \begin{cases} 1, n = 0 \text{ ou } n = 1 \\ n \,.\, (n-1)!, \text{se } n > 1 \end{cases}$$

Por exemplo, 4! = 4 . 3! = 4 . (3 . 2!) = (4 . 3) . (2 . 1!) = 4 . 3 . 2 . 1 = 24.

Naturalmente, uma definição recursiva necessita conter uma condição que interrompa a *recorrência*. Esta condição é chamada *condição de escape*. No caso do fatorial a condição de escape é n = 0 ou n = 1. A expressão que realiza propriamente a recorrência pode ser chamada *expressão de recorrência*

O que é surpreendente é que a maioria dos sistemas para desenvolvimento de programas permite que sejam definidas funções recursivas praticamente da mesma maneira que elas são escritas em matemática. Como o objetivo do lLA é apenas a aprendizagem da lógica de programação, os pesquisadores que o desenvolveram não incluíram recursos sofisticados de recursão. O ILA permite apenas a implementação de funções recursivas quando a expressão de recorrência contém uma única referência àquilo que se está definindo. Este é o caso, por exemplo, da definição recursiva do fatorial, cuja implementação em ILA é, simplesmente,

```
// Funcao recursiva que calcula o fatorial de um numero natural
Funcao FatRec(n)
Inicio
        Se (n = 0) ou (n = 1) entao
                Retornar 1
        Senao
                Retornar  n*FatRec(n-1)
        Fim_se
Fim
```

Se esta função for ativada para n = 4, por exemplo, ela *retorna* 4*FatRec(3). Isto significa que a função é ativada novamente para n = 3, ficando a expressão 4*FatRec(3) na *pilha de recursão*, aguardando o retorno de FatRec(3). Esta ativação *retorna* 3*FatRec(2) e a expressão na pilha, agora 4*3*FatRec(2), fica aguardando o *retorno* de FatRec(2), que é a expressão 2*FatRec(1). A expressão na tal *pilha de recursão* fica agora 4*3*2*FatRec(1), aguardando o *retorno* de FatRec(1). Como pela definição da função, FatRec(1) retorna 1, a expressão 4*3*2*1 é finalmente calculada e o seu valor, 24, é retornado para o comando que ativou a função para n = 4.

Devido à necessidade do empilhamento das expressões que calculam a função recursiva e das sucessivas ativações da mesma função, é fácil perceber que a utilização de uma função recursiva demanda um maior tempo computacional do que uma função não recursiva (chamada, geralmente, *função iterativa*). Assim, sempre que possível, devemos optar pela solução iterativa. No capítulo 7, porém, apresentaremos um exemplo (não executável) onde a solução recursiva seria tão eficiente quanto a iterativa, sendo bem mais elegante.

5.4 Exercícios propostos

1. Escreva uma função que retorne o k-ésimo dígito (da direita para esquerda) de um inteiro n, k e n dados. Por exemplo, K_esimoDigito(2845, 3) = 8.

2. O *fatorial ímpar* de um número n ímpar positivo é o produto de todos os números ímpares positivos menores do que ou iguais a n. Indicando o *fatorial ímpar* de n por n| temos, n| = 1 . 3. 5 . ... . n. Por exemplo, 7| = 1 . 3 . 5 . 7 = 105. Escreva funções, iterativa e recursiva, para a determinação do fatorial ímpar de um inteiro ímpar dado.

3. Como na questão anterior, o *fatorial primo* ou *primorial* de um número primo positivo é o produto de todos os primos positivos menores do que ou iguais a ele: p# = 2 . 3 . 5 . 7 . ... . p. Por

exemplo, 7# = 2 . 3 . 5 . 7 = 210. Escreva um programa que determine o primorial de um número primo dado.

4. Escreva funções, iterativa e recursiva, que retorne a soma dos algarismos de um inteiro positivo dado.

5. O exemplo 5 da seção 5.2, apresentou uma função que, sem usar funções pré-definidas, retornava a parte fracionária de um número positivo dado. Reescreva esta função sem a exigência da positividade do argumento.

6. Escreva uma função recursiva que determine o *mínimo múltiplo comum* de dois inteiros dados.

7. Escreva funções, recursiva e iterativa, que determinem b^n, b e n dados.

8. Escreva uma função recursiva que determine o *máximo divisor comum* de dois inteiros dados.