Notas de Aula de
Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083–970 Campinas–SP
{fkm,tomasz}@ic.unicamp.br

7 Procedimentos e Funções

A linguagem pascal permite que possamos declarar trechos de código fora do programa principal e associados a um identificador que podem ser executadas sempre que invocados. Chamaremos estes trechos de código por *módulos* ou *rotinas*.

Os principais motivos para se usar rotinas são:

1. Evitar codificação: trocar certos trechos de programas que se repetem por chamadas de apenas uma rotina que será codificada apenas uma vez.
2. Modularizar o programa, dividindo-o em módulos (rotinas) lógicamente coerentes, cada uma com função bem definida. Isto facilita a organização do programa, bem como o entendimento dele.

Na linguagem pascal, podemos declarar dois tipos de rotinas, que são os procedimentos e as funções.

7.1 Procedimentos

Um procedimento é sempre declarado em uma área de declarações, podendo ser tanto na área de declarações do programa principal como nas áreas de declarações de outras rotinas. Neste último caso, a *visibilidade* deste procedimento segue as regras de escopo (veja seção 7.4).

A forma geral de um procedimento segue o seguinte formato:

```
procedure Identificador_Procedimento(Lista_de_Parâmetros);
    Declarações   {Área de Declarações}
begin
    Comandos   {Corpo de Execução do Procedimento}
end;
```

O procedimento tem o mesmo formato do programa principal, sendo que o corpo do procedimento está delimitado por **begin** e **end**, este último terminado com um ponto e vírgula (Na rotina principal, o **end** é terminado com um ponto final). Além disso, a lista de parâmetros é opcional.

Na área de declarações, é possível definir constantes, tipos, declarar variáveis, funções e outros procedimentos, da mesma forma como declarados na área de declarações da rotina principal. É importante observar que os objetos declarados na área de declarações de uma rotina, são válidos apenas no escopo desta rotina, e não são válidos fora dela.

Exemplo 7.1 *Um determinado programa imprime o seguinte cabeçalho diversas vezes no meio de sua execução:*

```
Aluno: Fulano de Tal
Data: 01/01/01
Programa: Exemplo de Procedimento
```

Desta maneira, é melhor construir um procedimento que escreve este cabeçalho sempre que necessário. O pseudo-programa seguinte declara este procedimento e faz algumas chamadas no meio de sua execução.

```
program Exemplo;

procedure Cabecalho;
begin
    writeln('Aluno: Fulano de Tal.');
    writeln('Data: 01/01/01.');
    writeln('Programa: Exemplo de Procedimento.');
end;

begin
    ⋮
    Cabecalho;  {(*)}
    ⋮
    Cabecalho;  {(*)}
    ⋮
    Cabecalho;  {(*)}
    ⋮
end.
```

O cabeçalho será impresso nas chamadas da rotina Cabecalho, linhas comentadas com (*).

7.2 Passagem de Parâmetros

Como estratégia de programação, é desejável que cada rotina não faça uso de dados externos, pelo menos de forma direta, e toda a comunicação com o resto do programa seja feito através de parâmetros ou se for o caso, como retorno de função.

No programa seguinte, descrevemos um procedimento chamado *ImprimeMaximoDeTres* com três parâmetros reais: *x, y* e *z*. Os parâmetros declarados neste procedimento são declarados com *passagem de valor*. Isto significa que o procedimento ImprimeMaximoDeTres é chamado com três parâmetros:

ImprimeMaximoDeTres(Expressão 1, Expressão 2, Expressão 3);

onde Expressão 1, Expressão 2 e Expressão 3 são expressões numéricas que serão primeiro avaliadas para valores numéricos e só então transferidas para o procedimento. Internamente ao procedimento estes valores são acessados pelos parâmetros x, y e z.

```
program Maximos;
var a,b,c          : Real;
  i,n              : Integer;
procedure ImprimeMaximoDeTres(x,y,z : Real);
var t : Real;
begin
  t := x;
  if t>y then t := y;
  if t>z then t := z;
  writeln('O maiore entre ',x,',',y,',',z,' é ',t);
end; {MaximoDeTres}
begin
  Write('Entre com três números: ');
  Readln(a,b,c);
  ImprimeMaximoDeTres(a,b,c);
end.
```

Os parâmetros são declarados da seguinte forma:

[**var**]	ListaDeParâmetrosDoTipo1	Tipo1;
[**var**]	ListaDeParâmetrosDoTipo2	Tipo2;
⋮	⋮	⋮
[**var**]	ListaDeParâmetrosDoTipoK	TipoK;

onde [**var**] indica que a palavra **var** pode ou não ser colocada antes da lista de parâmetros de um tipo.

As palavras *Tipo1, Tipo2,..., TipoK* são palavras que identificam o tipo do parâmetro. Assim o seguinte cabeçalho de procedimento é inválido.

```
procedure Imprime(msg : string[50]);
```

Uma construção correta seria primeiro definir um tipo associado a **string[50]** e só então usar este tipo na declaração de *msg*. I.e.,

```
type TipoMSG = string[50];
procedure Imprime(msg : TipoMSG);
```

A inserção da palavra **var** antes da declaração de uma lista de parâmetros indica que estes parâmetros são declarados com Passagem por Referência e a ausência da palavra **var** indica que a lista de parâmetros seguinte é feita com Passagem por Valor.

A seguir, descrevemos os dois tipos de passagem de parâmetros.

Passagem por Valor Nesta forma, a expressão correspondênte ao parâmetro é avaliada e *apenas seu valor* é passado para o variável correspondente ao parâmetro dentro da rotina.

Passagem por Referência Nesta forma, o parâmetro que vai ser passado na chamada da rotina deve ser necessariamente uma va Isto porque não é o valor da variável que é passada no parâmetro, mas sim a sua referência. Qualquer alteração de valor no parâmetro correspondente refletirá em mudanças na variável correspondente, externa ao procedimento.

Exemplo 7.2 *No exemplo da figura 22 apresentamos um programa com uma rotina chamada Conta, contendo dois parâmetros, um por valor e outro por referência. Na linha (7), o comando de escrita imprime os valores de* $a = 20$

```
( 1)  Program exemplo;
( 2)  var x: integer;  y:real;
( 3)  Procedure Conta(a: integer; var b: real);
( 4)  begin
( 5)       a := 2 * a;
( 6)       b := 3 * b;
( 7)       writeln('O valor de a = ',a,' e o valor de b = ',b);
( 8)  end;
( 9)  begin
(10)       x := 10;
(11)       y := 30.0;
(12)       Conta(x, y);
(13)       writeln('O valor de x = ',x,' e o valor de y = ',y);
(14)  end.
```

Figura 22: Parâmetros por valor (a: integer) e por referência (var b: real).

e $b = 90.0$*. Na linha (13), depois da chamada da rotina Conta, são impressos os valores de* $x = 10$ *e* $y = 90.0$*, já que a passagem do parâmetro correspondente a* x *é por valor e a passagem do parâmetro correspondente a* y *é por referência (*x *tem o mesmo valor antes da chamada, e* y *tem o valor atualizado pela rotina Conta).*

Exemplo 7.3 *A seguir, apresentamos alguns exemplos do cabeçalho correspondente a declaração de rotinas com parâmetros:*

- *Procedure Conta(i, j:integer;var x, y:integer;var a, b:real);*
  Neste exemplo, os parâmetros correspondentes a i e j são declarações de parâmetros com passagem por valor; e x, y, a e b são declarações de parâmetros com passagem por referência.
- **type** *MeuTipoString = string[50];*
  Procedure processa_string(var nome1:MeuTipoString; nome2:MeuTipoString;var nome3:MeuTipoString; var a:integer; b:real);
  Neste exemplo, os parâmetros correspondentes a nome1, nome3 e a são declarações de parâmetros com passagem por referência; e nome2 e b são declarações de parâmetros com passagem por valor.

Exemplo 7.4 *O programa para ordenar três números, descrito na página 41, usa a estratégia de trocar os valores de variáveis. Note que naquele programa, a troca de valores é feita em três pontos do programa. Assim, nada melhor que fazer um procedimento para isso. O seguinte programa descreve esta alteração no programa ordena3. Note que os dois parâmetros da rotina trocaReal devem ser necessariamente declarados como parâmetros passados por referência.*

```
program ordena3;

procedure TrocaReal(var A, B : real);  {Troca os valores das duas variáveis}
var AUX: real;
begin
    AUX := A;
    A := B;
    B := AUX;
end;

var n1, n2, n3: real;
begin
    write('Entre com o primeiro número: ');
    readln(n1);
    write('Entre com o segundo número: ');
    readln(n2);
    write('Entre com o terceiro número: ');
    readln(n3);
    if (n1 > n2) then TrocaReal(n1, n2);
    if (n1 > n3) then TrocaReal(n1, n3);
    {Neste ponto, n1 contém o menor dos três valores}
    if (n2 > n3) then TrocaReal(n2, n3);
    { Neste ponto, n1 contém o menor dos três valores e n2 é menor ou igual a n3.}
    writeln(n1, n2, n3);
end.
```

Note que este programa ficou mais enxuto e mais fácil de entender.

Exemplo 7.5 *Um certo programa precisa ter, em vários pontos do seu código, leituras de números inteiros positivos e em cada um destes lugares, é necessário se fazer a validação do número lido. Uma maneira de se validar o número a ser lido é usar uma estrutura de repetição, como no exemplo 4.5 da página 33, em um procedimento que lê uma variável inteira e já faz sua validação. Desta maneira não precisaremos repetir o código para cada leitura.*

```
program ProgramaValidacao;
type messagetype = string[50];
procedure LeInteiroPositivo(var m : integer; msg:messagetype);
begin
  write(msg);
  readln(m);
  while (m<=0) do begin
    writeln('ERRO: Número inválido.');
    write(msg);
    readln(m);
  end;
end;

var n1,n2,n3:integer;
begin
  LeInteiroPositivo(n1,'Entre com o primeiro número inteiro positivo: ');
  LeInteiroPositivo(n2,'Entre com o segundo  número inteiro positivo: ');
  LeInteiroPositivo(n3,'Entre com o terceiro número inteiro positivo: ');
  writeln('Os três números positivos foram: ',n1,',',n2,',',n3);
end.
```

Consideração sobre a passagem de estruturas grandes como parâmetros

Muitas vezes, quando temos passagem de estruturas grandes (como vetores e matrizes) como parâmetros por valor, é preferível recodificar a rotina para que esta seja feita como parâmetro por referência. O motivo disto é justamente o fato destas estruturas serem duplicadas na passagem por valor. Na passagem de parâmetros por referência apenas a referência do objeto é transferida, gastando uma quantidade de memória constante para isso. Naturalmente esta codificação não deve mudar os valores da estrutura, caso contrário estas se manterão após o término da rotina.

Exemplo 7.6 *Considere os dois procedimentos seguintes, nas figuras 23 e 24, para imprimir o maior valor em um vetor V de n elementos. Note que nenhuma das duas rotinas faz alterações no vetor. A única diferença destas duas implementações é a passagem do parâmetro V por valor (figura 23) e por referência (figura 24). Assim, a implementação da figura 24 é mais eficiente que a da figura 23, caso o compilador em uso não faça nenhuma otimização para mudar o tipo da passagem de parâmetro.*

```
type TipoVetorReal = array[1..1000] of real;
procedure ImpMaximo(V : TipoVetorReal;
                    n : integer);
var M : real; i:integer;
begin
  if (n>0) then begin
    M := V[1];
    for i:=2 to n do
      if (V[i]>M) then M:=V[i];
  end
end.
```

Figura 23:

```
type TipoVetorReal = array[1..1000] of real;
procedure ImpMaximo(var V : TipoVetorReal;
                        n : integer);
var M : real; i:integer;
begin
  if (n>0) then begin
    M := V[1];
    for i:=2 to n do
      if (V[i]>M) then M:=V[i];
  end
end.
```

Figura 24:

7.3 Funções

A linguagem *Pascal* também permite desenvolver novas funções além das já existentes. Funções são rotinas parecidas com procedimentos, com a diferença que funções retornam um valor. Uma função é declarada com o seguinte cabeçalho:

function Identificador(Lista de Parâmetros): Tipo_da_Função;

O valor a ser retornado pela função é calculado dentro do *corpo* da função e para retorná-lo, é usado um identificador com o mesmo nome da função. O tipo de valor retornado é do tipo *Tipo_da_Função* que deve ser uma palavra identificadora do tipo. Assim, uma função não deve ser declarada como no exemplo a seguir

function Maiuscula(str: MeuTipoString):string[50];

Neste caso, **string[50]** não é apenas uma palavra. Assim, uma possível declaração seria:

function Maiuscula(str: MeuTipoString):MeuTipoString;

Exemplo 7.7 *No programa a seguir, temos a declaração de uma função chamada* **Cubo***, que dado um valor real (como parâmetro), a função devolve o cubo deste valor.*

```
program exemplo;
var a: real;
function Cubo(x:real): real;
begin
   Cubo:= x * x * x;
end;
begin
   write('Entre com o valor de a: ');
   readln(a);
   write('O cubo de ',a,' é ',Cubo(a));
end.
```

Exemplo 7.8 *No programa a seguir, apresentamos um programa com a declaração de uma função que devolve o maior valor entre dois valores, dados como parâmetros.*

```
program exemplo;
var a, b, c: real;
function Maximo(x,y: real): real;
begin
   if (x > y)
     then
        Maximo := x
     else
        Maximo := y;
end;
begin
   write('Entre com o valor de a: ');
   readln(a);
   write('Entre com o valor de b: ');
   readln(b);
   c := Maximo(a, b);
   write('O máximo entre ',a,' e ',b,' é: ',c);
end.
```

Exemplo 7.9 *Faça um programa contendo uma função que calcula o fatorial de um número passado como parâmetro inteiro.*

```
program ProgramaFatorial;
function fatorial(n:integer):integer;
var F,i:integer;
begin
  F := 1;
  for i:=2 to n do  F:=F * i;
  fatorial := F;
end;
var n:integer;
begin
  write('Entre com um número: ');  readln(n);
  writeln('O fatorial de ',n,' é igual a ',fatorial(n));
end.
```

Exemplo 7.10 *Implemente o algoritmo de Euclides para calcular o máximo divisor comum de dois números.*

```
program ProgramaMDC;
function mdc(a,b : integer):integer;
var aux,maior,menor : integer;
begin
  maior := a;  menor := b;
  while (menor <> 0) do begin
    aux := menor;
    menor := maior mod menor;
    maior := aux;
  end;
  mdc := maior;
end;
var a,b:integer;
begin
  write('Entre com dois numeros positivos: '); readln(a,b);
  writeln('O mdc dos dois é ',mdc(a,b));
end.
```

Exemplo 7.11 *Descreva uma função para testar se um número é primo usando a estratégia do programa Primo2 apresentada na página 49.*

```
function Primo(n :integer ):boolean;
var i,Max    : integer;
begin
  if (n = 2) then Primo := true
  else if (n mod 2 = 0) or (n<=1) then Primo:= False
  else begin
    Primo := True;
    i:=3;
    Max := trunc(sqrt(n));
    while (i<= Max) and (n mod i <>0) do i := i+2;
    if (i<=Max) and (n mod i = 0) then Primo := false
  end
end
```

7.4 Escopo

Todos os objetos declarados em um programa (subprogramas, variáveis, constantes, tipos, etc) possuem um *escopo* de atuação. Por escopo de um objeto, entendemos como as regiões de um programa onde o objeto é válido.

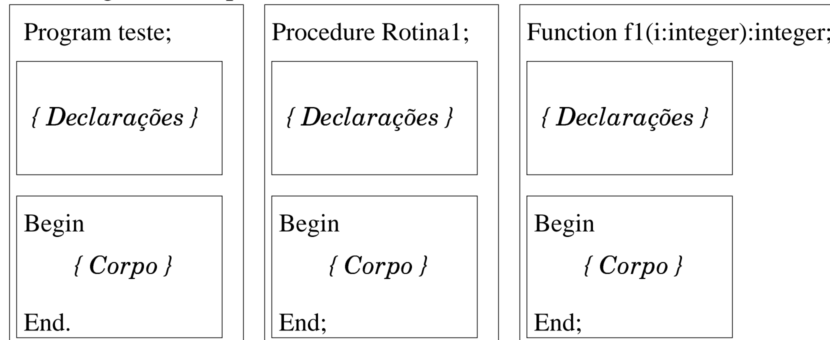
Primeiramente, vamos definir alguns termos:

Cabeçalho de uma Rotina É o texto que indica o tipo da rotina (se programa principal, procedimento ou função), seu nome e seus parâmetros.

Área de Declaração da Rotina Área onde se declaram as variáveis, tipos, procedimentos e funções da rotina.

Corpo de uma Rotina Definiremos o Corpo de uma Rotina (pode ser o Programa Principal, um Procedimento ou uma Função) como o trecho da rotina contendo as instruções a serem executadas.

Na figura a seguir, temos alguns exemplos de rotinas.



Note também que na área de declaração de uma rotina podemos declarar novas rotinas. Na figura 25, temos uma estrutura de um programa com várias rotinas, declaradas uma dentro da área de declarações da outra.

Se em algum lugar é feito alguma referência a um objeto do programa, este já deve ter sido declarado em alguma posição acima desta referência.

Os objetos declarados em uma rotina **R** são visíveis no corpo de **R** e em todas as subrotinas dentro da área de declarações de **R**. Por visualizar, queremos dizer que podemos usar a variável/procedimento/função no local de visualização.

Se dois objetos têm o mesmo nome, a referência através deste nome é feita para o objeto que estiver na área de declarações visível mais interna. Exemplo: considere duas rotinas, **R** e **R1**, onde **R1** está declarado na área de declarações de **R** (**R1** está contido em **R**). Um identificador declarado dentro de **R1** pode ser declarado com o mesmo nome de um outro identificador em **R** (externo a R1). Neste caso, a variável declarada em **R** não será visualizada em R1.

Na figura 25 apresentamos o esboço de um programa com diversos níveis de encaixamento de rotinas. A seguir, descrevemos quais objetos podem ser visualizados em cada região:

1. Na região 1, podemos *visualizar* os objetos: **C, B, A** (primeiro A), **Rotina1** e **Rotina2**.
2. Na região 2, podemos *visualizar* os objetos: **D, B, A** (segundo A), **Função1, Rotina1** e **Rotina2**.
3. Na região 3, podemos *visualizar* os objetos: **A** (primeiro A), **B, Função1, Rotina1** e **Rotina2**.
4. Na região 4, podemos *visualizar* os objetos: **E, A** (primeiro A) e **Rotina3** e **Rotina1**.
5. Na região 5, podemos *visualizar* os objetos: **A** (primeiro A) e **Rotina1** e **Rotina3**.

Exemplo 7.12 *O programa seguinte apresenta um procedimento que encontra as raizes reais de uma equação do segundo grau. O procedimento se chama Equacao, tem três parâmetros e chama uma função particular chamado CalculaDelta. Por ser particular a este procedimento, a função CalculaDelta pode estar dentro da área de declarações*

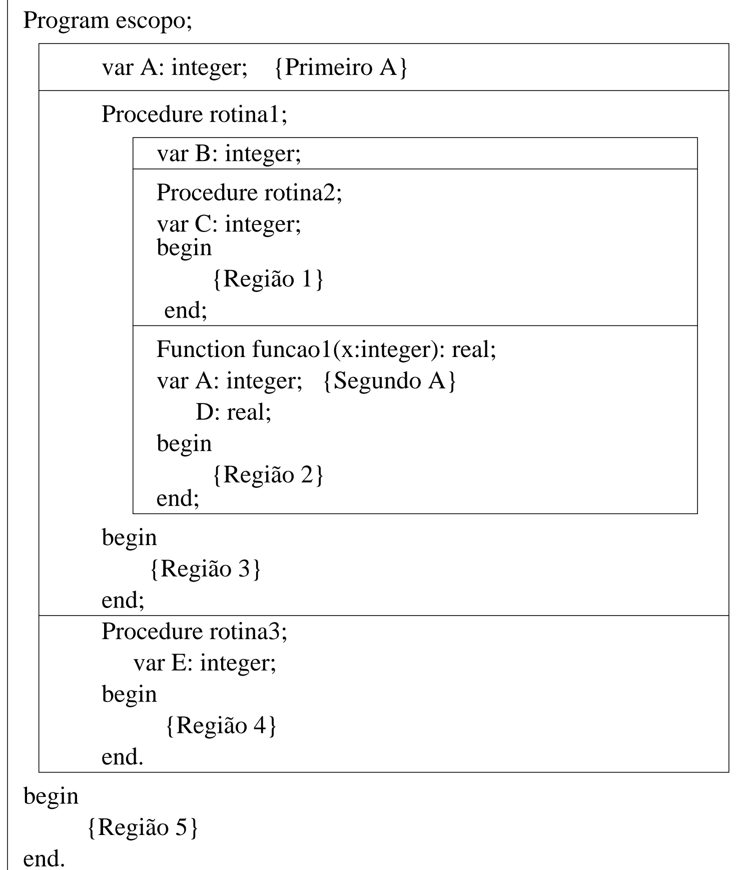


Figura 25: Exemplo de objetos com escopos diferentes.

de Equacao.

```
program EquacaoSegundoGrau;
var a_ext, b_ext, c_ext : real;
procedure Equacao(a,b,c: real); { imprime as soluções reais de a∗x∗x + b∗x+c = 0 }
var x1, x2,delta              : real; { Variáveis locais ao procedimento.}
function CalculaDelta(c1,c2,c3: real): real;
begin
  CalculaDelta := sqr(c2) − 4∗c1∗c3;
end;
begin
  delta:=CalculaDelta(a,b,c);
  if (delta>=0) then begin
    x1:=(−b+sqrt(delta)) / (2∗a);     x2:=(−b−sqrt(delta)) / (2∗a);
    writeln('O valor de x1 = ',x1,' e o valor de x2 = ',x2);
  end else writeln('não é possivel calcular raizes reais para esta equação');
end;
begin
  writeln('Encontrar raizes reais de equação na forma: a*x*x + b*x+c = 0');
  write('Entre com o valor de a (diferente de zero), b e c: '); readln(a_ext,b_ext,c_ext);
  Equacao(a_ext,b_ext,c_ext);
end.
```

Exemplo 7.13 *O seguinte programa apresenta um procedimento para ordenação usando a estratégia do algoritmo SelectionSort, descrita com duas subrotinas encaixadas.*

```
program SelectionSort;
const MAX        =  100;
type TipoVetorReal = array[1..MAX] of real;
  TipoMsg        =  string[100];
procedure LeVetorReal(var v : TipoVetorReal; n:integer;msg:TipoMsg);
var i : integer;
begin
  writeln(msg);
  for i:=1 to n do begin writeln('Entre com o ',i,'-ésimo elemento: '); readln(V[i]); end
end; { LeVetorReal }
procedure ImprimeVetorReal(var v : TipoVetorReal; n:integer;msg:TipoMsg);
var i : integer;
begin
  writeln(msg);  for i:=1 to n do begin writeln(V[i]); end
end; { ImprimeVetorReal }
procedure SelectionSort(var v : TipoVetorReal; n:integer);
var m,imax : integer;
  procedure TrocaReal(var a,b : real);
  var aux : real;
  begin   aux := a;    a:=b;    b:=aux;   end;
  function IndMaximo(var v : TipoVetorReal; n : integer) : integer;
  var i, Ind : integer;
  begin
    if (n <=0) then Ind := 0
    else begin
      Ind := 1; {O maior elemento começa com o primeiro elemento do vetor}
      for i:=2 to n do if v[Ind] < v[i] then Ind:=i;
    end;
    IndMaximo := Ind;
  end;
begin
  for m:=n downto 2 do begin
    imax := IndMaximo(v,m);
    TrocaReal(v[m],v[imax]);
  end;
end; { SelectionSort }
var i, n, Ind : integer;
  V       : TipoVetorReal;
begin
  write('Entre com a quantidade de elementos a ler: '); readln(n);
  LeVetorReal(v,n,'Leitura do vetor a ordenar');
  SelectionSort(v,n);
  ImprimeVetorReal(v,n,'Vetor ordenado');
end.
```

7.5 Cuidados na Modularização de Programas

A modularização dos programas pode levar ao desenvolvimento de programas mais indepententes e mais fáceis de se entender. Mas há casos onde devemos tomar algum cuidado, se quisermos evitar processamentos desnecessários. Dois casos onde podemos ter processamento desnecessário são:

1. Rotinas distintas fazendo os mesmos cálculos, i.e., alguns dados vão ser recalculados.
2. Chamadas de uma mesma rotina várias vezes, onde uma chamada pode ter feito cálculos já realizados pela chamada anterior.

Muitas vezes priorizamos a independência e funcionalidade das rotinas e toleramos que algumas computações sejam duplicados. Isto é razoável quando estas duplicações não são críticas no tempo de processamento total. Quando o tempo de processamento deve ser priorizado, devemos reconsiderar o programa e evitar estas computações desnecessárias. O seguinte exemplo mostra duas versões de um programa para calcular a exponencial (e^x).

Exemplo 7.14 *A exponencial (e^x) pode ser calculada pela seguinte série:*

$$e^x = x^0 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

O programa seguinte mostra uma implementação da função exponencial, usando duas outras funções, uma para calcular o fatorial, e outra para calcular a potência. O programa pára quando a diferença entre o valor da série calculado até uma interação com o valor calculado na interação anterior é menor que 0,00001.

```
program exprog1;
var   y : real;
function fatorial(n : integer):integer;
var i,f      : integer;
begin
  i:=1;  f:=1;
  while (i<=n) do begin     f:=f*i;  i:=i+1;  end;
  fatorial:=f;
end;
function potencia(x : real; n:integer):real;
var i : integer;
   p : real;
begin
  p:=1;
  for i:=1 to n do p:=p*x;
  potencia := p;
end;
function expo1(x : real):real;
var termo,ex,exant: real;  i:integer;
begin
  ex := 0;  termo:=1;  i := 0;
  repeat
    exant := ex;
    termo := potencia(x,i)/fatorial(i);
    ex := ex + termo;
    inc(i);
  until abs(ex−exant)<0.0001;
  expo1 := ex;
end;
begin
  write('Entre com um número real: ');  readln(y);
  writeln('O valor de e elevado a  ',y,' é igual a ',expo1(y));
end.
```

O programa exprog1 é fácil de entender, uma vez que cada termo da série é facilmente calculado atráves da função fatorial e da função potência. Mas note que ao se calcular um termo genérico desta série, em uma interação, digamos o termo $\frac{x^k}{k!}$, o programa calcula o valor x^k, sendo que em interações anteriores, já foram computados os valores de x^1, x^2,..., x^{k-1}. Cada um destes cálculos poderia ter aproveitado o cálculo da potência feita na interação anterior. O mesmo acontece com o fatorial, $k!$, que poderia ter aproveitado o cálculo de $(k-1)!$ feito na interação anterior. Portanto, o termo calculado em uma interação é igual ao termo anterior, $\frac{x^{k-1}}{(k-1)!}$, multiplicado por $\frac{x}{k}$. O seguinte programa apresenta a versão modificada, sem repetição de processamento.

```
program exprog2;
var   y : real;
function expo2(x : real):real;
var termo,ex,exant : real;  i:integer;
begin
  termo:=1;  i := 1;  ex := 0;
  repeat
    exant := ex;
    ex := ex + termo;
    termo := termo * x/i;
    inc(i);
  until abs(ex−exant)<0.0001;
  expo2 := ex;
end;
begin
  write('Entre com um número real: ');
  readln(y);
  writeln('O valor de e elevado a  ',y,' é igual a ',expo2(y));
end.
```

Exercício 7.1 *O valor do seno(x) pode ser dado pela seguinte série:*

$$\text{seno}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Implemente uma função chamada seno(x), para calcular o valor do seno de x. Note que x^{k+2} é igual a $x^k * x^2$ e $(n+2)!$ é igual a $n! \cdot (n+1) \cdot (n+2)$.
OBS.: *A função deve usar no máximo um loop. I.e, não se pode usar loops encaixados.*

Exercício 7.2 *O valor do co-seno de x pode ser calculado pela serie*

$$co\text{−}seno(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

Descreva uma função com o cabeçalho: function coseno(x:real):real;
e que devolve o valor do coseno de x calculado com os 100 primeiros termos dada pela série acima.
OBS.: *A função deve usar no máximo um loop. I.e, não se pode usar loops encaixados.*

7.6 Exercícios

1. Faça três versões da função fatorial:

   function fatorial(n:**integer**):**integer**;

   usando as estruturas de repetição: **for**, **while** e **repeat**.
2. Os números de fibonacci $n_1, n_2, \ldots$ são definidos da seguinte forma:

$$\begin{aligned} n_0 &= 0, \\ n_1 &= 1, \\ n_i &= n_{i-1} + n_{i-2}, \quad i \geq 2. \end{aligned}$$

Faça um programa contendo uma função

function fibonacci(n:**integer**):**integer**;

que retorna o n-ésimo número de fibonacci.

3. Faça um procedimento com um parâmetro inteiro n e que ao ser chamado, o procedimento imprime uma figura da seguinte forma:

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

No caso, o procedimento foi chamado com parâmetro 5. A quantidade de linhas impressas é $2n - 1$.

4. O valor de π também pode ser calculado usando a série $S = \frac{1}{1^3} - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \frac{1}{9^3} - \frac{1}{11^3} + \cdots$, sendo que o valor de π é calculado como $\pi = \sqrt[3]{32 \cdot S}$. Faça uma função para calcular o valor de π usando esta série e parando quando a diferença do valor de π calculado em uma interação e a interação anterior for menor que 0,0001.
5. Escreva um programa contendo uma função

function binario(n:integer):TipoString;

que retorna um tipo chamado *TipoString*, declarado como

type TipoString = string[50];

string contendo o número n na base binária.

6. Faça duas funções que tem um parâmetro inteiro e retornam verdadeiro se o parâmetro for primo e falso caso contrário, uma função usando o método ingênuo apresentado na página 48 e o método otimizado apresentado na página 49. Verifique quanto tempo estas duas funções gastam se usadas para contar todos os primos no intervalo $[1000, 5000]$. Experimente para outros intervalos maiores.
7. Implemente conjuntos através de vetores e faça os seguintes procedimentos:

   (a) Procedimento chamado conjunto, com dois parâmetros, um vetor e um inteiro positivo $n \geq 0$, indicando a quantidade de elementos no vetor. O procedimento deve remover os elementos duplicados do vetor e atualizar o valor de n.

   (b) Procedimento chamado *Intersecao* com seis parâmetros, com o seguinte cabeçalho:
   procedure *intersecao*(**var** v_1:tipovetor;n_1:integer;
   var v_2:tipovetor;n_2:integer;
   var v_3:tipovetor; **var** n_3:integer);
   O vetor v_1 (v_2) contém n_1 (n_2) elementos. O vetor v_3 receberá a interseção dos elementos de v_1 e v_2 e n_3 deve retornar com a quantidade de elementos em v_3 (vamos supor que cada vetor contém elementos distintos).
   Você deve usar a seguinte estratégia para gerar a interseção de v_1 e v_2:
   (1) Ordene o vetor v_1.
   (2) Para cada elemento de v_2, faça uma busca binária do elemento no vetor v_1.
   (2.1) Se o elemento se encontrar no vetor, insira o elemento no vetor v_3.

   (c) Procedimento chamado *Uniao* com seis parâmetros, com o seguinte cabeçalho:
   procedure *uniao*(**var** v_1:tipovetor;n_1:integer;
   var v_2:tipovetor;n_2:integer;
   var v_3:tipovetor; **var** n_3:integer);
   O vetor v_1 (v_2) contém n_1 (n_2) elementos. O vetor v_3 receberá a união dos elementos de v_1 e v_2 e n_3 deve retornar com a quantidade de elementos em v_3 (vamos supor que cada vetor contém elementos distintos).
   Você deve usar a seguinte estratégia para gerar a união de v_1 e v_2:
   (1) Copie o vetor v_1 no vetor v_3 (atualizando n_3).
   (2) Para cada elemento de v_2, faça uma busca binária do elemento no vetor v_1.
   (2.1) Se o elemento não se encontrar no vetor, insira o elemento no vetor v_3.

8. Tem-se um conjunto de dados contendo a altura e o sexo (M ou F) de 50 pessoas. Fazer um algoritmo que calcule e escreva:

   (a) A maior e a menor altura do grupo.
   (b) A média de altura das mulheres.
   (c) O número de homens.

9. Descreva uma função que tenha como parâmetros uma razão, r, um valor inicial, v_0, e um número n e devolva a soma dos n primeiros elementos de uma progressão aritmética começando com v_0 e com razão r.
10. Descreva um procedimento, com parâmetro inteiro positivo n e dois outros parâmetros b e k que devem retornar valores inteiros positivos. Os valores a serem retornados em b e k são tais que b seja o menor valor inteiro tal que $b^k = n$.
11. Faça uma função booleana com parâmetro n e que retorna verdadeiro se n é primo, falso caso contrário.
12. Faça uma função que tenha como parâmetro uma temperatura em graus Fahrenheit e retorne a temperatura em graus Celsius. Obs.: $(C = 5/9 \cdot (F - 32))$.
13. O imposto que um trabalhador paga depende da sua faixa salarial. Existem até k faixas salariais, cada uma com uma correspondente taxa. Exemplo de um sistema com até 4 faixas salariais:

   (a) Para salários entre 0 e 100 reais, é livre de imposto.
   (b) Para salários maiores que 100 e até 500 reais, é 10 % de imposto.
   (c) Para salários maiores que 500 e até 2000 reais, é 20 % de imposto.
   (d) Para salários maiores que 2000 é 30 % de imposto.

   Faça um programa que leia estas k faixas salariais e leia uma seqüência de salários e imprima para cada um, o imposto a pagar. O programa deve parar quando for dado um salário de valor negativo.
14. Faça um programa de lotérica, que lê o nome de n jogadores e os números que eles apostaram (um número entre 0 e 100). Use a função RANDOM(N) para sortear um número. Se houver ganhador, imprima o nome dele e o número que ele apostou, caso contrário, avise que ninguém ganhou.