

## Continuação do Programa 7.7

```

procedure LiberaGrafo (var Grafo: TipoGrafo);
begin
  { Nao faz nada no caso de posicoes contiguas }
end; { LiberaGrafo }

procedure ImprimeGrafo (var Grafo: TipoGrafo);
var i: integer;
begin
  writeln('      Cab Prox Peso');
  for i := 0 to Grafo.NumVertices+2*Grafo.NumArestas-1 do
    writeln(i:2, Grafo.Cab[i]:4, Grafo.Prox[i]:4, Grafo.Peso[i]:4);
  end; { ImprimeGrafo }

```

## 7.2.4 Programa Teste para as Três Implementações

O programa para testar os operadores do tipo abstrato de dados pode ser visto no Programa 7.8. Qualquer uma das três implementações apresentadas nas Seções 7.2.1, 7.2.2 ou 7.2.3 pode ser usada com este programa. Observe que o procedimento que implementa o operador *InserereAresta* pode ser usado para criar grafos direcionados ou não direcionados. A inserção de uma aresta contendo os vértices  $v$  e  $u$  em um grafo não direcionado pode ser realizada por meio de duas chamadas de *InserereAresta*, uma para a aresta  $(v, u)$  e outra para a aresta  $(u, v)$ , como ilustra o Programa 7.8.

## Programa 7.8 Programa teste para operadores do tipo abstrato de dados grafo

```

program TestaOperadoresTADGrafo;

{-- Entra aqui a estrutura do tipo grafo do Programa 7.2 --}
{-- ou do Programa 7.4 ou do Programa 7.6 --}

var
  Aux      : TipoApontador;
  i        : integer;
  V1, V2, Adj : TipoValorVertice;
  Peso     : TipoPeso;
  Grafo, Grafot: TipoGrafo;
  NVertices  : TipoValorVertice;
  NArestas   : 0..MAXNUMARESTAS;
  FimListaAdj : boolean;

{-- Entram aqui os operadores correspondentes do Programa 7.3 --}
{-- ou do Programa 7.5 ou do Programa 7.7 --}

```

## Continuação do Programa 7.8

```

begin {-- Programa principal --}
{-- NumVertices: definido antes da leitura das arestas --}
{-- NumArestas: inicializado com zero e incrementado a --}
{-- cada chamada de InserereAresta --}
  write ('No. vertices: '); readln (NVertices);
  write ('No. arestas: '); readln (NArestas);
  Grafo.NumVertices := NVertices; Grafo.NumArestas := 0;
  FGVazio (Grafo);
  for i := 0 to NArestas - 1 do
    begin
      write ('Inserere V1 -- V2 -- Peso: '); readln (V1, V2, Peso);
      Grafo.NumArestas := Grafo.NumArestas + 1;
      InserereAresta (V1, V2, Peso, Grafo); { 1 chamada g-direcionado }
      InserereAresta (V2, V1, Peso, Grafo); { 2 chamadas g-naodirecionado }
    end;
  ImprimeGrafo (Grafo); readln;
  write ('Inserere V1 -- V2 -- Peso: '); readln (V1, V2, Peso);
  if ExisteAresta (V1, V2, Grafo)
  then writeln ('Aresta ja existe')
  else begin
    Grafo.NumArestas := Grafo.NumArestas + 1;
    InserereAresta (V1, V2, Peso, Grafo);
    InserereAresta (V2, V1, Peso, Grafo);
  end;
  ImprimeGrafo (Grafo); readln;
  write ('Lista adjacentes de: '); read (V1);
  if not ListaAdjVazia (V1, Grafo)
  then begin
    Aux := PrimeiroListaAdj (V1, Grafo); FimListaAdj := false;
    while not FimListaAdj do
      begin
        ProxAdj (V1, Grafo, Adj, Peso, Aux, FimListaAdj);
        write (Adj:2, ' (', Peso, ')');
        end;
        writeln; readln;
      end;
      write ('Retira aresta V1 -- V2: '); readln (V1, V2);
      if ExisteAresta (V1, V2, Grafo)
      then begin
        Grafo.NumArestas := Grafo.NumArestas - 1;
        RetiraAresta (V1, V2, Peso, Grafo);
        RetiraAresta (V2, V1, Peso, Grafo);
      end
      else writeln ('Aresta nao existe');
      ImprimeGrafo (Grafo); readln;
      write ('Existe aresta V1 -- V2: '); readln (V1, V2);
      if ExisteAresta (V1, V2, Grafo) then writeln ('Sim') else writeln ('Nao');
      LiberaGrafo (Grafo);
    end.

```