Programa 7.4 *Estrutura do tipo grafo implementado como listas encadeadas usando apontadores*

```
const MAXNUMVERTICES = 100;
      MAXNUMARESTAS  = 4500;
type
  TipoValorVertice = 0..MAXNUMVERTICES;
  TipoPeso         = integer;
  TipoItem         = record
                       Vertice: TipoValorVertice;
                       Peso   : TipoPeso;
                     end;
  TipoApontador    = ^TipoCelula;
  TipoCelula       = record
                       Item: TipoItem;
                       Prox: TipoApontador;
                     end;
  TipoLista        = record
                       Primeiro: TipoApontador;
                       Ultimo: TipoApontador;
                     end;
  TipoGrafo        = record
                       Adj         : array[TipoValorVertice] of TipoLista;
                       NumVertices: TipoValorVertice;
                       NumArestas : 0..MAXNUMARESTAS;
                     end;
```

Programa 7.5 *Operadores sobre grafos implementados como listas de adjacência usando apontadores*

```
{-- Entram aqui os operadores FLVazia, Vazia e Insere do Programa 3.4 --}

procedure FGVazio (var Grafo: TipoGrafo);
var i: integer;
begin
  for i := 0 to Grafo.NumVertices-1 do FLVazia (Grafo.Adj[i]);
end; { FGVazio }

procedure InsereAresta (V1, V2   : TipoValorVertice;
                        Peso     : TipoPeso;
                        var Grafo: TipoGrafo);
var x    : TipoItem;
begin
  x.Vertice := V2;
  x.Peso    := Peso;
  Insere (x, Grafo.Adj[V1]);
end; { InsereAresta }
```


Continuação do Programa 7.5

```
function ExisteAresta (Vertice1, Vertice2: TipoValorVertice;
                       var Grafo: TipoGrafo): boolean;
var Aux: TipoApontador;  EncontrouAresta: boolean;
begin
  Aux := Grafo.Adj[Vertice1].Primeiro^.Prox;
  EncontrouAresta := false;
  while (Aux <> nil) and (EncontrouAresta = false) do
    begin
    if Vertice2 = Aux^.Item.Vertice then EncontrouAresta := true;
    Aux := Aux^.Prox;
    end;
  ExisteAresta := EncontrouAresta;
end; { ExisteAresta }

{-- Operadores para obter a lista de adjacentes --}
function ListaAdjVazia (Vertice: TipoValorVertice;
                        var Grafo: TipoGrafo): boolean;
begin
  ListaAdjVazia := Grafo.Adj[Vertice].Primeiro =
                   Grafo.Adj[Vertice].Ultimo;
end; { ListaAdjVazia }

function PrimeiroListaAdj (Vertice: TipoValorVertice;
                           var Grafo: TipoGrafo): TipoApontador;
begin
  PrimeiroListaAdj := Grafo.Adj[Vertice].Primeiro^.Prox;
end; { PrimeiroListaAdj }

procedure ProxAdj (Vertice        : TipoValorVertice;
                   var Grafo      : TipoGrafo;
                   var Adj        : TipoValorVertice;
                   var Peso       : TipoPeso;
                   var Prox       : TipoApontador;
                   var FimListaAdj : boolean);
{--Retorna Adj e Peso do Item apontado por Prox--}
begin
  Adj     := Prox^.Item.Vertice;
  Peso    := Prox^.Item.Peso;
  Prox    := Prox^.Prox;
  if Prox = nil then FimListaAdj := true;
end; { ProxAdj- }

procedure RetiraAresta (V1, V2: TipoValorVertice;
                        var Peso : TipoPeso; var Grafo : TipoGrafo);
var AuxAnterior, Aux: TipoApontador;
    EncontrouAresta: boolean; x: TipoItem;

{-- Entra aqui o operador Retira do Programa 3.4 --}
```