**Programa G.5** *Operadores implementados como listas de adjacência usando apontadores*

```
/*-- Entram aqui os operadores FLVazia, Vazia --*/
/*-- e Insere do Programa C.4                 --*/

void FGVazio(TipoGrafo *Grafo)
{ long i;
  for (i = 0; i < Grafo->NumVertices; i++) FLVazia(&Grafo->Adj[i]);
}

void InsereAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ TipoItem x;
  x.Vertice = *V2;
  x.Peso = *Peso;
  Insere(&x, &Grafo->Adj[*V1]);
}

short ExisteAresta(TipoValorVertice Vertice1,
                   TipoValorVertice Vertice2,
                   TipoGrafo *Grafo)
{ TipoApontador Aux;
  short EncontrouAresta = FALSE;
  Aux = Grafo->Adj[Vertice1].Primeiro->Prox;
  while (Aux != NULL && EncontrouAresta == FALSE)
    { if (Vertice2 == Aux->Item.Vertice) EncontrouAresta = TRUE;
      Aux = Aux->Prox;
    }
  return EncontrouAresta;
}

/* Operadores para obter a lista de adjacentes */
short ListaAdjVazia(TipoValorVertice *Vertice, TipoGrafo *Grafo)
{ return (Grafo->Adj[*Vertice].Primeiro ==
          Grafo->Adj[*Vertice].Ultimo);
}

TipoApontador PrimeiroListaAdj(TipoValorVertice *Vertice,
                               TipoGrafo *Grafo)
{ return (Grafo->Adj[*Vertice].Primeiro->Prox); }

void ProxAdj(TipoValorVertice *Vertice, TipoGrafo *Grafo,
             TipoValorVertice *Adj, TipoPeso *Peso,
             TipoApontador *Prox, short *FimListaAdj)
{ /* Retorna Adj e Peso do Item apontado por Prox */
  *Adj = (*Prox)->Item.Vertice;
  *Peso = (*Prox)->Item.Peso;
  *Prox = (*Prox)->Prox;
  if (*Prox == NULL) *FimListaAdj = TRUE;
}
```



*Continuação do Programa G.5*

```
/*-- Entra aqui o operador Retira do Programa C.4 --*/
void RetiraAresta(TipoValorVertice *V1, TipoValorVertice *V2,
                  TipoPeso *Peso, TipoGrafo *Grafo)
{ TipoApontador AuxAnterior, Aux;
  short EncontrouAresta = FALSE;
  TipoItem x;
  AuxAnterior = Grafo->Adj[*V1].Primeiro;
  Aux = Grafo->Adj[*V1].Primeiro->Prox;
  while (Aux != NULL && EncontrouAresta == FALSE)
    { if (*V2 == Aux->Item.Vertice)
      { Retira(AuxAnterior, &Grafo->Adj[*V1], &x);
        Grafo->NumArestas--;
        EncontrouAresta = TRUE;
      }
      AuxAnterior = Aux;
      Aux = Aux->Prox;
    }
}

void LiberaGrafo(TipoGrafo *Grafo)
{ TipoApontador AuxAnterior, Aux;
  for (i = 0; i < GRAfo->NumVertices; i++)
    { Aux = Grafo->Adj[i].Primeiro->Prox;
      free(Grafo->Adj[i].Primeiro);    /*Libera celula cabeca*/
      Grafo->Adj[i].Primeiro=NULL;
      while (Aux != NULL)
        { AuxAnterior = Aux;
          Aux = Aux->Prox;
          free(AuxAnterior);
        }
    }
  Grafo->NumVertices = 0;
}

void ImprimeGrafo(TipoGrafo *Grafo)
{ int i;
  TipoApontador Aux;
  for (i = 0; i < Grafo->NumVertices; i++)
    { printf("Vertice%2d:", i);
      if (!Vazia(Grafo->Adj[i]))
      { Aux = Grafo->Adj[i].Primeiro->Prox;
        while (Aux != NULL)
          { printf("%3d (%d)", Aux->Item.Vertice, Aux->Item.Peso);
            Aux = Aux->Prox;
          }
      }
      putchar('\n');
    }
}
```