```
 1: #include <malloc.h>
 2: #include <stdlib.h>
 3:
 4: #include "VertexVectorAdjacencyList.h"
 5:
 6: VertexVectorAdjacencyList::VertexVectorAdjacencyList ( )
 7: {
 8:     m_lastHighestDegree = 0;
 9:     m_nVertex = 0;
10: }
11:
12: VertexVectorAdjacencyList::~VertexVectorAdjacencyList ( )
13: {
14:     delete m_elementList;
15:
16:     for(int i = 0; i <  m_nVertex - 1; i++) {
17:         delete m_vectorVertex[i];
18:     }
19:
20:     delete m_vectorVertex;
21: }
22:
23: void VertexVectorAdjacencyList::Allocate( int nVertex )
24: {
25:     AdjacencyList::Allocate( nVertex );
26:
27:     m_nVertex = nVertex;
28:
29:     m_lastHighestDegree = nVertex - 2;
30:     m_vectorVertex = new List * [ nVertex - 1 ];
31:
32:     for(int i = 0; i <  nVertex - 1; i++) {
33:         m_vectorVertex[i] = new List();
34:     }
35:
36:     m_elementList = new ListNode *[nVertex];
37:
38:     for(int i = 0; i < nVertex; i++) {
39:         m_elementList[i] = NULL;
40:     }
41:
42: }
43:
44: void VertexVectorAdjacencyList::DecrementDegree( int iVertex )
45: {
46:     // remove da lista atual e ..
47:     ListNode* element  = m_elementList[ iVertex ];
48:     int degree = element->getDegree();
49:
50:     m_vectorVertex[ degree ]->remove( element );
51:
52:     // insere na lista respectiva ao grau-1
53:     ListNode* node = m_vectorVertex[ degree - 1 ]->insertAtEnd( iVertex );
54:
55:     node->setDegree( degree - 1 );
56:
57:     m_elementList[ iVertex ] = node;
58: }
59:
60: void VertexVectorAdjacencyList::RemoveFromVertexVector( int iVertex, int iDegree )
61: {
62:     ListNode* element  = m_elementList[ iVertex ];
63:
64:     m_vectorVertex[ iDegree ]->remove( element );
65:     m_elementList[ iVertex ] = NULL;
66: }
67:
68: int VertexVectorAdjacencyList::RemoveHighestDegreeVertex( int debug )
69: {
70:     ListNode * highestDegreeVertex = GetHighestDegreeVertex();
71:     int iHighestDegreeVertex = highestDegreeVertex->getVertex();
72:     List * neighbors = m_arrAdjLists[ iHighestDegreeVertex ];
73:
74:     if ( debug >= 2 )
75:     {
76:         fprintf( stderr, "  vertice %d tem %d vizinhos\n",
77:                  iHighestDegreeVertex,
78:                  neighbors->size( ) );
79:     }
80:
81:     for ( ListNode * node = neighbors->getFirst();
82:             node != NULL;
83:             node = node->next())
84:     {
85:         int iNeighbor = node->getVertex();
86:         //int iCurrentDegree = (int) m_arrAdjLists[ iNeighbor ]->size();
87:
88:         // update this vertex's neighbor's list that this vertex is being removed
89:         //m_arrAdjLists[ iNeighbor ]->erase( iHighestDegreeVertex );
90:
91:         // if the neighbor is still in vertex vector, it means it has not been removed
92:         // note: it may be on my neighbor's list, but already processed and removed, we need
93:         // to make sure we will be decrementing a degree from a neighbor that's already in the graph
94:         if ( m_elementList[ iNeighbor ] != NULL )
```

```
 95:         {
 96:             // remove edge from this vertex
 97:             m_nEdges --;
 98:
 99:             // decrement degree from neighbor
100:             DecrementDegree( iNeighbor );//, iCurrentDegree );
101:         }
102:     }
103:
104:
105:     // reset degree
106:     RemoveFromVertexVector( iHighestDegreeVertex, highestDegreeVertex->getDegree() );//neighbors->size() );
107:
108:     // remove edges to neighbors, and let the vertex linger and ...
109:     //neighbors.clear( );
110:
111:     return iHighestDegreeVertex;
112: }
113:
114: ListNode* VertexVectorAdjacencyList::GetHighestDegreeVertex( )
115: {
116:     for(int i = m_lastHighestDegree; i >= 0; i--)
117:     {
118:         if(m_vectorVertex[i]->size() != 0)
119:         {
120:             m_lastHighestDegree = i;
121:             return m_vectorVertex[i]->getFirst();//)->getVertex();
122:         }
123:     }
124:
125:     return NULL;
126: }
127:
128: void VertexVectorAdjacencyList::updateData()
129: {
130:     for( int i = 0; i < m_nVertex; i++ )
131:     {
132:         int degree = m_arrAdjLists[i]->size();
133:
134:         // the vertex degree is its position in the vector
135:         ListNode* node = m_vectorVertex[ degree ]->insertAtEnd( i );
136:         node->setDegree( degree );
137:
138:         m_elementList[i] = node;
139:     }
140: }
```