### Class Graph

#### Constructors

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>graph()</code></td>
<td>Initializes a graph of type T with 0 vertices and 0 edges.</td>
</tr>
<tr>
<td><code>graph(const graph&lt;T&gt;&amp;)</code></td>
<td>Copy constructor – duplicate existing graph of type T.</td>
</tr>
</tbody>
</table>

#### Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>graph&lt;T&gt;&amp; operator=(const graph&lt;T&gt;&amp; rhs)</code></td>
<td>Assignment operator</td>
</tr>
<tr>
<td><code>map&lt;T, vertexInfo&lt;T&gt;&gt;::iterator getIter(const T&amp; v)</code></td>
<td>Returns an iterator pointing to the provided vertex.</td>
</tr>
<tr>
<td><code>bool inGraph(const T&amp; v)</code></td>
<td>Returns whether a vertex is in the graph.</td>
</tr>
<tr>
<td><code>vertexInfo&lt;T&gt;&amp; getInfo(const T&amp; v)</code></td>
<td>Get the information associated with a given vertex. Returns a <code>vertexInfo</code> object.</td>
</tr>
<tr>
<td><code>typename map&lt;T, vertexInfo&lt;T&gt;&gt;::iterator begin()</code></td>
<td>Returns a map iterator pointing to the first element of the graph.</td>
</tr>
<tr>
<td><code>typename map&lt;T, vertexInfo&lt;T&gt;&gt;::iterator end()</code></td>
<td>Returns a map iterator pointing to the last element of the graph.</td>
</tr>
<tr>
<td><code>int numberOfVertices() const</code></td>
<td>Return the number of vertices in the graph.</td>
</tr>
<tr>
<td><code>int numberOfEdges() const</code></td>
<td>Return the number of edges in the graph.</td>
</tr>
<tr>
<td><code>bool empty() const</code></td>
<td>Returns True if the graph is empty, otherwise returns False.</td>
</tr>
<tr>
<td><code>int inDegree(const T&amp; v)</code></td>
<td>Returns the number of connections going into vertex v. Returns -1 if the vertex does not exist.</td>
</tr>
<tr>
<td><code>int outDegree(const T&amp; v)</code></td>
<td>Returns the number of connections going out of vertex v. Returns -1 if the vertex does not exist.</td>
</tr>
</tbody>
</table>
set<T> getNeighbors(const T& v);
    Returns the set of neighbors of vertex v.

T operator[](const T& v);
    Return the “first” object from the map

bool insertEdge(const T& v1, const T& v2);
    Create an edge between v1 and v2, return true if successful, false otherwise

bool insertVertex(const T& v);
    Add a new vertex (v) to the graph, return true if successful, false otherwise

bool eraseEdge(const T& v1, const T& v2);
    Remove an edge between v1 and v2, return true if successful, false otherwise

bool eraseVertex(const T& v);
    Remove a vertex from the map, return true if successful, false otherwise

void clear();
    Remove all vertices and edges from the graph

---

**CLASS vertexInfo**  **Constructors**  “graph.h”
vertexInfo(); inDegree(0);
    Initializes vertexInfo object, sets inDegree to 0

---

**CLASS Peg**  **Constructors**  “peg.h”
Peg(int x = 0, int y = 0, pegColor = RED, bool = false, bool = false);
    Default constructor, creates new peg object

Peg(const Peg&);
    Copy constructor

~Peg();
    Deconstructor

---

**CLASS Peg**  **Operations**  “peg.h”
bool onFirstEdge() const;
    Return true if peg is on the first (left or upper) edge of the board
bool onSecondEdge() const;
    Return true if peg is on the second (right or lower) edge of the board

int getX() const;
    Return the x coordinate of the peg

int getY() const;
    Return the y coordinate of the peg

pegColor getColor() const;
    Return the color of the peg

Peg& operator=(const Peg&);
    Overloaded assignment operator

bool operator==(const Peg&) const;
    Overloaded equality operator – necessary for insertion into graph

bool operator<(const Peg&) const;
    Overloaded less than operator – necessary for insertion into graph

<table>
<thead>
<tr>
<th>CLASS Board</th>
<th>Constructors</th>
<th>“board.h”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board(int, int);</td>
<td>Default constructor, accepts and sets max x and max y values, initializes vector</td>
<td></td>
</tr>
<tr>
<td>~Board();</td>
<td>Deconstructor</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS Board</th>
<th>Operations</th>
<th>“board.h”</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool isPeg(int x, int y);</td>
<td>Return true if there is a peg at (x,y)</td>
<td></td>
</tr>
<tr>
<td>bool validConnect(int x1, int y1, int x2, int y2);</td>
<td>Return true if a connection is possible between (x1,y1) and (x2,y2)</td>
<td></td>
</tr>
<tr>
<td>bool addPeg(int x, int y, Peg::pegColor col);</td>
<td>Add a peg at (x,y), return true if successful</td>
<td></td>
</tr>
<tr>
<td>bool addConnect(int x1, int y1, int x2, int y2, Peg::pegColor);</td>
<td>Connect the pegs at (x1,y1) and (x2,y2), return true if successful</td>
<td></td>
</tr>
<tr>
<td>bool removePeg(int x, int y, Peg::pegColor);</td>
<td>Remove the peg at (x,y), return true if successful</td>
<td></td>
</tr>
</tbody>
</table>
bool removeConnect(int x1, int y1, int x2, int y2, Peg::pegColor);
    Remove the connection between (x1,y1) and (x2,y2), return true if successful

void drawBoard();
    Refresh the display of the board

void drawPeg(int x, int y);
    Draw a peg at (x,y)

void drawConnect(int x1, int y1, int x2, int y2);
    Draw a connection between (x1,y1) and (x2, y2)

**CLASS boardInfo**

Constructor:
boardInfo(Peg::pegColor,);
    Default constructor – creates a new boardInfo object with no connections

Destructor:
~boardInfo();
    Deconstructor

**CLASS boardInfo**

Operation:
bool hasConnection(int x, int y);
    Return if this object has a connection to a peg at (x,y)

**CLASS Player**

Constructor:
Player(int, int, playerColor);
    Default constructor – initializes new player with board max values and a color

Destructor:
~Player();
    Deconstructor

**CLASS Player**

Operation:
void setBoard(Board&);
    Set the player’s board pointer to a provided board object

void takeTurn();
    Have the player perform a move

bool addPeg(int x, int y);
    Add a peg to the player’s board at (x,y), return true if successful

bool connectPegs(int x1, int y1, int x2, int y2);
    Connect pegs (x1,y1) and (x2,y2) on the player board, return true if successful
bool removePeg(int x, int y);
    Remove a peg from the player’s board, return true if successful

bool removeConnect(int x1, int y1, int x2, int y2);
    Disconnect pegs (x1,y1) and (x2,y2), return true if successful

playerColor getColor();
    Return the color of the player

bool won();
    Return true if the player has met the conditions to win

bool quit();
    Return true if the player has resigned

void setDimensions(int, int, int, int, int, int);
    openGL function to establish size of the display window

void myMouseFunc(int, int, int, int);
    openGL function to enable use of mouse