```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "List.h"
#include "Graph.h"

/*** Private Function Prototypes ***/
int insertEdge( ListRef L, int u );
int isInOrderRange( int u, int order );
void killGraph( char* e );

/*** Graph Constructor / Destructor ***/
GraphRef newGraph( int n ){
   int i;
   GraphRef G = malloc( sizeof(Graph) );
   assert( G != NULL );
   G->order = n;
   G->size = G->source = NIL;

   G->adj = malloc( (n+1) * sizeof(ListRef*) );
   assert( G->adj != NULL );

   G->color = calloc( (n+1),  sizeof(int) );
   assert( G->color != NULL );

   G->d = calloc( (n+1) , sizeof(int) );
   assert( G->d != NULL );

   G->P = calloc( n+1, sizeof(int) );
   assert( G->P != NULL );

   for ( i = 1; i <= n; i++ ){
      G->adj[i] = newList();
      G->d[i] = INF;
   }
   return G;
}

void freeGraph( GraphRef* pG ){
   int i;
   if (pG != NULL && *pG != NULL ){
      for ( i = 1; i <= getOrder((*pG)); i++ ){
         freeList( &((*pG)->adj[i]) );
      }
      free((*pG)->adj);
```

```c
         free((*pG)->color);
         free((*pG)->d);
         free((*pG)->P);
         free(*pG);
         *pG = NULL;
      }
   }

   /*** Access Functions ***/
   int getOrder( GraphRef G ) {
      if ( G == NULL ) killGraph("Calling getOrder() on NULL GraphRef");
      return G->order;
   }

   int getSize( GraphRef G ) {
      if ( G == NULL ) killGraph("Calling getSize() on NULL GraphRef");
      return G->size;
   }

   int getSource( GraphRef G ) {
      if ( G == NULL ) killGraph("Calling getSource() on NULL GraphRef");
      if ( G->source == NIL ) return NIL;
      else return G->source;
   }

   int getParent( GraphRef G, int u ){
      if ( G == NULL ) killGraph("Calling getParent() on NULL GraphRef");
      if ( !isInOrderRange(u, getOrder(G) ) )
         killGraph("Method getParent() requires an input vertex u \
   such that 1 <= u <= Order of graph");

      if ( G->source == NIL ) return NIL;
      else return G->P[u];
   }

   int getDist( GraphRef G, int u ){
      if ( G == NULL ) killGraph("Calling getDist() on NULL GraphRef");
      if ( !isInOrderRange(u, getOrder(G) ) )
         killGraph("Method getDist() requires an input vertex u \
   such that 1 <= u <= Order of graph");
      if ( G->source == NIL ) return INF;
      else return G->d[u];
   }

   void getPath( ListRef L, GraphRef G, int u ) {
```

```
   if ( G == NULL ) killGraph("Calling getPath() on NULL GraphRef");
   if ( !isInOrderRange(u, getOrder(G) ) )
      killGraph("Method getPath() requires an input vertex u \
such that 1 <= u <= Order of graph");
   /*append NIL to list if no path exists*/
   if ( G->source == u ){
      insertFront(L,u);
   } else if ( G->P[u] == NIL ){
      insertFront(L,NIL); /*path doesn't exist*/
   } else {
      getPath(L,G,G->P[u]);
      insertBack(L,u);
   }
}



/*** Manipulation Procedures ***/
void makeNull( GraphRef G ){
    if ( G == NULL ) killGraph("Calling addEdge() on NULL GraphRef");
   int i;
   for ( i = 1; i <= getOrder(G); i++ ){
      makeEmpty(G->adj[i]);
      G->d[i] = INF;
      G->P[i] = G->color[i] = NIL;
   }
   G->size = G->source = 0;
}

void addEdge( GraphRef G, int u, int v ){
   if ( G == NULL ) killGraph("Calling addEdge() on NULL GraphRef");
   if ( !isInOrderRange(u, getOrder(G) ) ||
            !isInOrderRange(v, getOrder(G) ) )
      killGraph("Method addEdge() requires precondition u and v within\
range of 1 to the order of G");

   ListRef uList = G->adj[u];
   ListRef vList = G->adj[v];

   if ( insertEdge( vList, u ) && insertEdge( uList, v) )
      G->size += 1;

}

void addArc( GraphRef G, int u, int v ){
```

```
   if ( G == NULL ) killGraph("Calling addArc() on NULL GraphRef");
   if ( !isInOrderRange(u, getOrder(G) ) ||
            !isInOrderRange(v, getOrder(G) ) )
      killGraph("Method addArc() requires precondition u and v within\
range of 1 to the order of G");

   if ( insertEdge(G->adj[u], v) ) G->size += 1;
}

void BFS( GraphRef G, int s ){
   int i, x, y;
   ListRef Q, L;
   G->source = s;
   for ( i = 1; i <= getOrder(G); i++ ){
      if ( i != s) {
         G->color[i] = WHITE;
         G->d[i] = INF;
         G->P[i] = NIL;
      }
   }
   G->color[s] = GREY;
   G->d[s] = G->P[s] = NIL;
   /* Q = FIFO queue, where enqueue = insertback, dequeue = delete front */
   Q = newList();
   insertBack(Q,s);
   while ( !isEmpty(Q) ){
      x = getFront(Q);
      deleteFront(Q);
      L = G->adj[x];
      moveTo(L,0);
      while( !offEnd(L) ){
         y = getCurrent(L);
         if ( G->color[y] == WHITE ){
            G->color[y] = GREY;
            G->d[y] = G->d[x]+1;
            G->P[y] = x;
            insertBack(Q,y);
         }
         moveNext(L);
      }
      G->color[x] = BLACK;
   }
   freeList(&Q);
}
```

```c
/* insertEdge() - adds u to the adjacency list L in sorted order,
   returns 0 if edge already exists in adj list, 1 otherwise.
*/
int insertEdge( ListRef L, int u ) {
   if ( isEmpty(L) ) {
      insertFront(L, u);
   }else {
      moveTo(L,0);
      int entry = getCurrent(L);
      while ( entry < u ){
         moveNext(L);
         if ( offEnd(L) ) { entry = NIL; break; }
         else entry = getCurrent(L);
      }
      if ( entry == NIL ) {
         insertBack(L,u);
      } else if ( entry > u ){
         insertBeforeCurrent(L,u);
      }else return 0;
      /* the only other case is that this edge already exists, do nothing */
   }
   return 1;
}

/*** Other functions ***/
void printGraph( FILE* out, GraphRef G ){
   if ( G == NULL ) killGraph("Calling printGraph() on NULL GraphRef");
   int i;
   for ( i = 1; i <= getOrder(G); i++ ){
      if ( !isEmpty( G->adj[i] ) ){
         fprintf( out, "%d:", i);
         moveTo(G->adj[i],0);
         while( !offEnd(G->adj[i]) ){
            fprintf(out, " %d", getCurrent( G->adj[i]) );
            moveNext( G->adj[i] );
         }
         fprintf( out, "\n" );
      }
   }
}

/* killGraph() - prints error e to stdout and exits program */
void killGraph( char* e ) {
   printf( "Graph.c: %s\n", e);
   exit(1);
```

```c
}

void printGraphInfo( GraphRef G ){
   if ( G == NULL ) killGraph("Calling printGraphInfo() on NULL GraphRef");
   printf("Graph G has\nsize %d\norder %d \
            \nsource %d\n",getSize(G),getOrder(G),getSource(G));
}

/* isInOrderRange() - returns true if 1 <= u <= order. false otherwise. */
int isInOrderRange( int u, int order ){
   if ( u < 1 || u > order) return 0;
   else return 1;
}
```