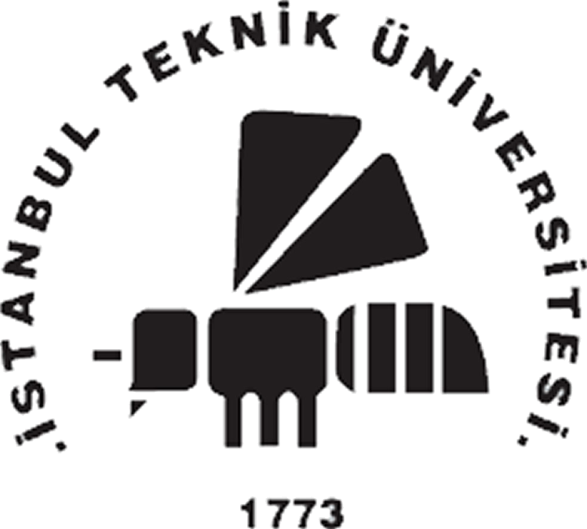
**I.T.U.**

**Faculty of Electric-Electronic**

**Computer Engineering**



Lesson name: Object Oriented Programming

Lesson Code: BLG252E

Name Surname: Aykut Akın

Number: 040080177

Instructor’s Name: Feza Buzluca

Due Date: 01.05.2010

**Introduction**

I implemented classes that can be used to read and represent undirected graphs, directed graphs, trees in C++.

My classes represent graphs using its name and an array of nodes. Also each node in a graph

hold references to the adjacent nodes in the graph.

In my homework there exists a has-a relationship between a graph class and node class. Each

node may exist in a single graph but a graph can contain many exposing a one-to-many relationship.

A graph can be created in one of the three ways.

– An empty graph that doesn't contain any node or edge

– A graph with a predetermined size and arbitrary node names, but no connections.

– A graph as a copy of other

These operations can be made for graph

– A node can be added to a graph. There can't be two nodes with same name in a graph.

– A node can be deleted from a graph, together with its connections.

– An edge can be added/deleted from a graph. An edge can be added/deleted only if its nodes

exist in the graph.

– The graph can be output to screen

– Two graphs can be intersected. The resulting graph contains the intersection of both nodes and

edges.

– Two graphs can be unified. The resulting graph contains the union of both nodes and edges.

Trees and directed graphs is derived from Graph class and expose all the operations of graphs

except the following restrictions:

-A directed graph can only be united and intersected with another directed graph. Otherwise

there should be a compiler error.

– A tree can only be united and intersected with another tree. Otherwise there should be a

compiler error.

– Directed graph's and tree's addEdge and deleteEdge operations are quite different from

Graph's. It only adds directional adjacents.

– A tree doesn't have addEdge or deleteEdge operations, instead it has addChild and

deleteChild which also adds directional adjacents to nodes.

**Explanation of Classes**

* **Explanation of Graph.h**

name: Name of the graph

nodes: Nodes which are included in the graph

nofNodes: Number of nodes included in the graph

containsNode(string): Query if a node is present

opBuf: Operations buffer

graphFile: Pointer for file operations

nOfGraphs: Total graph count

Graph(): Constructor

Graph(int): Constructor which takes a number of nodes as a parameter

Graph(string): Constructor which takes the name of file as a parameter

Graph(const Graph &): Copy constructor

~Graph(): Destructor

getNumOfNodes(): A function that helps us to learn number of nodes

getGraphName(): A function that helps us to learn the name of a graph

addNode(string): A function that adds a new node to the graph and takes a name of node as a

parameter. Returns true if the node is added

deleteNode(string): A function that deletes a node from the graph and takes the name of the

node as a parameter. Returns null if the node is absent

addEdge(string,string): A function that adds a new edge between two nodes and takes the names

of the nodes as a parameter. Returns true if the edge is added

deleteEdge(string,string): A function that deletes an edge between two nodes and takes the name

of the nodes as a parameter. Returns false if the edge is absent

intersect(const Graph &): A function that intersect the two graphs and takes a graph as a

parameter. Prints a new intersection graph to the screen

unite(const Graph &):A function that unites the two graphs and takes a graph as a

parameter. Prints a new union graph to the screen

toString: A function that print the graph to the screen

acceptTraverse(BreadthFirst\*): A function that helps us to traverse with a breadth first method.

* **Explanation of Node.h**

name: Name of the node

adjacents: Pointer to adjacent of node

nofAdjacents: Number of adjacent

containsAdjacent(string): If contains adjacent in <par> return its indice

Node(): Constructor

Node(string): Constructor which takes a name of the node as a parameter

Node(const Node &): Copy constructor

~Node(): Destructor

getNodeName(): A function that helps us to learn the name of the node

toString: Get string representation of Node

getAdjacent(int): A function that returns the adjacent of node

getNumOfAdjacents(): A function that returns the number of adjacent

addAdjacent(const Node &): A function that adds a new adjacent to the node and takes the

node as a parameter

deleteAdjacent(const Node&): A function that deletes the adjacent of the node

intersect(const Node&): Intersect node adjacent with the node in the <par>

unite(const Node&): Unite node adjacent with the node in the <par>

* **Explanation of Tree.h**

addEdge(string, string): This is in private part of Tree class. So you can not use this function directly.

deleteEdge(string, string) : This is in private part of Tree class. So you can not use this function

directly.

Tree(): Constructor

Tree(int): Constructor which takes a number of nodes as a parameter

Tree(string): Constructor which takes the name of file as a parameter

Tree(const Tree &):Copy constructor

~Tree():Destructor

intersect(const Tree&): A function that intersect the two trees and takes a tree as a

parameter. Prints a new intersection tree to the screen

unite(const Tree&): A function that unites the two trees and takes a tree as a

parameter. Prints a new union tree to the screen

addChild(string, string):Add a child to the tree

deleteChild(string, string):Delete a child from the tree

* **Explanation of Directed.h**

Directed():Constructor

Directed(int): Constructor which takes a number of nodes as a parameter

Directed(string):Constructor which takes the name of file as a parameter

Directed(const Directed &): Copy constructor

~Directed():Destructor

intersect(const Directed&): A function that intersect the two directed graphs and takes a directed

graph as a parameter. Prints a new intersection directed graph to the screen

unite(const Directed&): A function that unite the two directed graphs and takes a directed graph as a parameter. Prints a new unique directed graph to the screen

addEdge(string, string): Add an edge to the directed graph

deleteEdge(string, string):Delete an edge from the directed graph

**Missing parts of the homework**

In my homework traversal part is missing. So you can not work with traversal methods. But still there exists Traversal.h, DepthFirst.h, BreadthFirst.h. I am going to explain these classes but the functions’ bodies are empty.

* **Explanation of Traversal.h**

Traversal(): Constructor

~Traversal(): Destructor

visited: Visited nodes during traversal

(virtual )traverse()=0: An abstract function to helps us not to create an object from this class

* **Explanation of BreadthFirst.h**

BreadFirst(): Constructor

~BreadthFirst(): Destructor

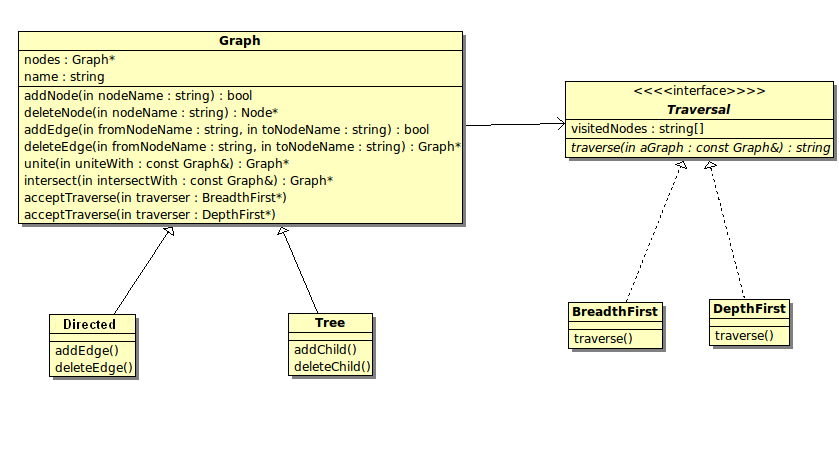
traverse(): A function that helps us traverse with breadth first method

* **Explanation of DepthFirst.h**

DepthFirst(): Constructor

~DepthFirst(): Destructor

traverse(): A function that helps us traverse with depth first method

**Association Between Classes**