**Object Oriented Programming 2009-2010 Spring Term**
**1st Homework Assignment**

Due On: 14.03.2010 23:00[1]

You are going to implement classes that can be used to read and represent undirected graphs in C++. Your classes should represent graphs using its name and an array of nodes. Also each node in a graph should hold references to the adjacent nodes in the graph.

Please pay attention that 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 four ways.

- Having its content from a file (see Appendix B)
- 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.

Following the definitions above, your class implementations should be based on the following interfaces:

```
class Node {
  private:
    string name;
    Node** adjacent;
  public:
    Node();
    Node(string);
    Node(const Node &);
    ~Node();
    bool addAdjacent(const Node &);
    Node** getAdjacents();
    string getName();
}
```

1 Due date is strict, e-mail submissions will be subject to penalty. Be cautious and submit a version before last minute rush.

```
class Graph {
  private:
    int numNodes,numEdges;
    string name;
    Node* nodes;
    fstream* graphFile;
  public:
    static int graphCount;
    Graph();
    Graph(string filename);
    Graph(int);
    Graph(const Graph &);
    ~Graph();
    bool newNode(string);              //returns true if the node is added
    bool newEdge(string,string);       //returns true if the edge is added
    Node* deleteNode(string);          //returns null if the node is absent
    bool deleteEdge(string,string);   //returns false if the edge is absent
    string printGraph();
    Graph* intersect(const Graph &);  //returns a new intersection graph.
    Graph* unite(const Graph &);     //returns a new union graph.
}
```

**Pay attention:**

- You should extend the class definitions above, also you may perform minimal changes if required.
- Node arrays must be hold dynamic
- Each graph must have a graph file attached to it(even though it wasn't declared explicitly) during its lifetime. Every operation on the graph must be logged into the file. You should use a static variable to generate different file names for each graph. (see Appendix A and B)
- The program must be written using object oriented principles. Try to use C++ built-in classes as much as possible. Using C++ string class instead of char arrays may be very useful, check the following link for information: http://www.cplusplus.com/reference/string/string/
- Academic dishonesty including but not limited to cheating, plagiarism, collaboration is unacceptable and subject to disciplinary actions. Any student found guilty will have -200 penalty points.

**Evaluation Criteria:**

- Programming(70 Points) :
  - Compilation: Your program must compile under g++. Programs that doesn't compile will be subject to a penalty of 20 points.
  - Usage of OO Principles(55 Points):
    - Class/object usage (15 Points)
    - Constructor destructor usage (20 Points)
    - Property/method usage and encapsulation (20 Points)
  - General programming(10 points): Your program shouldn't have any memory leaks, poor programming discipline .You should declare each class in a seperate (.h - .cpp) pair. Including Makefile is appreciated.
  - Program Comprehension/Comments(5 Points): You should at least include a few words for important methods.

- Execution(20 Points) : Programs that produce no output or doesn't work at all receive 0-5 points; programs that work erroneously or by giving segmentation faults receive 5-15 points; programs that work without problem receive 15-20 points

- Report(10 Points) : **Programs without reports will not be evaluated.** A good report should include explanations regarding “general software structure” “program workflow” “important classes and data structures” “important methods” in a structured fashion. A good report shall explain the most with fewest words, remember that “A picture is worth a thousand words”. A good report shall not include copy-paste code blocks or comments.

**How to submit:**

- You should archive your report(.pdf), program files(.cpp, .h, Makefile etc.) and input files(graph files for this assignment) in a .zip or .rar file named after your student number (e.g. 504052505.rar)
- Submit your work to ninova. Submit a working copy at least half an hour before the due time to avoid last minute rush accidents.
- For any questions-comments please contact only with R.A. Tolga Ovatman.

**Appendix A – C++ File operations**

Data output operations on text files are performed in the same way we operated with `cout`:

```
1
2  // writing on a text file
3  #include <iostream>
4  #include <fstream>
5  using namespace std;
6
7  int main () {
8    ofstream myfile ("example.txt");
9    if (myfile.is_open())
10   {
11     myfile << "This is a line.\n";
12     myfile << "This is another line.\n";
13     myfile.close();
14   }
15   else cout << "Unable to open file";
16   return 0;
17 }
18
19
```

```
[file example.txt]
This is a line.
This is another line.
```

Data input from a file can also be performed in the same way that we did with `cin`:

```
1
2
3  // reading a text file
4  #include <iostream>
5  #include <fstream>
6  #include <string>
7  using namespace std;
8
9  int main () {
10   string line;
11   ifstream myfile ("example.txt");
12   if (myfile.is_open())
13   {
14     while (! myfile.eof() )
15     {
16       getline (myfile,line);
17       cout << line << endl;
18     }
19     myfile.close();
20   }
21
22   else cout << "Unable to open file";
23
24   return 0;
25 }
26
27
28
```

```
This is a line.
This is another line.
```

This last example reads a text file and prints out its content on the screen. Notice how we have used a new member function, called `eof()` that returns true in the case that the end of the file has been reached. We have created a while loop that finishes when indeed `myfile.eof()` becomes true (i.e., the end of the file has been reached).

## Checking state flags

In addition to `eof()`, which checks if the end of file has been reached, other member functions exist to check the state of a stream (all of them return a bool value):

bad()
: Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

fail()
: Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

eof()
: Returns true if a file open for reading has reached the end.

good()
: It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

In order to reset the state flags checked by any of these member functions we have just seen we can use the member function `clear()`, which takes no parameters.

More information can be obtained from: http://www.cplusplus.com/doc/tutorial/files/

**Appendix B – Graph log file example**

```
graph aGraph
A B C D E F
A-B
B-C
A-D
C-F
C-D
D-E
E-F
-------------------------------------
intersected with anotherGraph
edge A-D has been deleted
node A has been deleted
edge A-B has been deleted
united with myGraph
aGraph deleted
```