```cpp
#include "graph.h"
#include <algorithm>
#include <array>
#include <cassert>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
using namespace std;

graph::graph(const string &file_name)
: _edges(0), _nvert(0)
{
    ifstream fin(file_name);            // Oeffne das File im ASCII-Modus
    if ( fin.is_open() ) {              // File gefunden:
//        _edges.clear();                 // Vektor leeren
        unsigned int k,l;
        while ( fin >> k >> l) _edges.push_back({k,l}); // Einlesen
        if (!fin.eof()) {
            // Fehlerbehandlung
            cout << " Error handling \n";
            if ( fin.bad() )  throw runtime_error("Schwerer Fehler in istr");
            if ( fin.fail() ) { // Versuch des Aufraeumens
                cout << " Failed in reading all data.\n";
                fin.clear();
            }
        }
        _edges.shrink_to_fit();
    }
    else {                              // File nicht gefunden:
        cout << "\nFile " << file_name << " has not been found.\n\n";
        assert( fin.is_open() && "File not found." );       // exeption handling for the poor programmer
    }

    DetermineNumberVertices();

    return;
}


vector<vector<unsigned int>> graph::get_node2nodes() const
{
    size_t nnode=Nvertices();

    // Determine the neighborhood for each vertex
    vector<vector<unsigned int>> n2n(nnode);
    for (size_t k=0; k<_edges.size(); ++k)
    {
        const int v0 = _edges[k][0];
        const int v1 = _edges[k][1];
        n2n.at(v0).push_back(v1);         // add v1 to neighborhood of v0
```

```
   53:             n2n.at(v1).push_back(v0);          //     and vice versa
   54:         }
   55:         // ascending sort of entries per node
   56:         for (size_t k=0; k<n2n.size(); ++k)
   57:         {
   58:             sort(n2n[k].begin(),n2n[k].end());
   59:         }
   60:
   61:
   62:         return n2n;
   63: }
   64:
   65:  void graph::DetermineNumberVertices()
   66:  {
   67:      // we assume that the nodes are numbered consecutively from 0 to
 n-1
   68:         // determine number of nodes
   69:         unsigned int nnode=0;
   70:         for (size_t k=0; k<_edges.size(); ++k)
   71:         {
   72:             for (size_t j=0; j<_edges[k].size(); ++j)
   73:             {
   74:                 nnode=max(nnode,_edges[k][j]);
   75:             }
   76:         }
   77:         if (_edges.size()>0) ++nnode;          // more than 1 edge i grap
h?
   78:         _nvert = nnode;
   79:  }
   80:
   81: ostream& operator<<(ostream &s, graph const &rhs)
   82: {
   83:         auto &edges=rhs._edges;
   84:         s << "\n --  Edges  --\n";
   85:         for (size_t k=0; k<edges.size(); ++k)
   86:         {
   87:             s << k << " : ";
   88:             for (size_t j=0; j<edges[k].size(); ++j)
   89:             {
   90:                 s << edges[k][j] << " ";
   91:             }
   92:             s << endl;
   93:         }
   94:
   95:         s << "Graph with  " << rhs.Nedges() << " edges and  " << rhs.Nver
tices() << " vertices" << endl;
   96:
   97:         return s;
   98: }
   99:
```

```
    1: #pragma once
    2:
    3: #include <array>
    4: #include <iostream>
    5: #include <string>
    6: #include <vector>
    7:
    8: /**
    9:   A simple graph class that requires a consecutive numbering of the v
ertices starting from 0.
   10: */
   11: class graph {
   12: public:
   13:     /** \brief Reads edges for graph from file.
   14:      *
   15:      *  If the file  @p file_name does not exist then the code stops
with an appropriate message.
   16:      *
   17:      *  A consecutive numbering of the vertices is required.
   18:      *
   19:      * @param[in]    file_name   name of the ASCII-file
   20:      */
   21:     graph(const std::string &file_name);
   22:
   23:     graph(graph const & org) = default;
   24:     graph& operator=(graph const & rhs) = default;
   25:
   26:     /**
   27:       Determines the neighboring vertices for each node from the edge
 definition.
   28:       The node itself is not contained in the neighboring vertices.
   29:
   30:       @return       vector[nn][*] with all neighboring vertices for e
ach node
   31:     */
   32:     std::vector<std::vector<unsigned int>> get_node2nodes() const;
   33:
   34:     /**
   35:       @return       number of edges
   36:     */
   37:     size_t Nedges() const
   38:     {
   39:         return _edges.size();
   40:     }
   41:
   42:     /**
   43:       @return       number of vertices
   44:     */
   45:     size_t Nvertices() const
   46:     {
   47:         return _nvert;
   48:     }
   49:
   50:     friend std::ostream& operator<<(std::ostream &s, graph const &rhs
);
   51:
   52: private:
```

```
   53:     /**
   54:       Determines the number of vertices from the edge information.
   55:       It requires a consecutive numbering of the vertices starting fr
om 0.
   56:     */
   57:     void DetermineNumberVertices();
   58:
   59:     std::vector<std::array<unsigned int, 2>> _edges;  /**< stores the
 two vertices for each edge */
   60:     size_t                                   _nvert;  /**< number of
vertices */
   61:
   62: };
   63:
```

```cpp
//graph
#include "graph.h"
#include <array>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main()
{
    cout << "Hello Graph!" << endl;
    const graph g1{"g_1.txt"};

    cout << g1 << endl;

    // construct mapping nodes to nodes
    auto n2n=g1.get_node2nodes();

    cout << "\n --  Nodes to Node  --\n";
    for (size_t k=0; k<n2n.size(); ++k)
    {
        cout << k << " : ";
        for (size_t j=0; j<n2n[k].size(); ++j)
        {
            cout << n2n[k].at(j) << " ";
        }
        cout << endl;
    }

    return 0;
}
```