```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), _vertices(), _maxvert(-1)             // graph_2
{
    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." );       // exception handling for the poor programmer
    }

    DetermineNumberVertices();

    return;
}


vector<vector<unsigned int>> graph::get_node2nodes() const
{
//    size_t nnode=Nvertices();
    size_t nnode=Max_vertex()+1;          // graph_2

    // 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 o
f v0
        n2n.at(v1).push_back(v0);          //      and vice versa
    }
    // ascending sort of entries per node
    for (size_t k=0; k<n2n.size(); ++k)
    {
        sort(n2n[k].begin(),n2n[k].end());
    }


    return n2n;
}
            // graph_2
 void graph::DetermineNumberVertices()
 {
     // we assume that the nodes are numbered consecutively from 0 to n-1
    // determine number of nodes
    _vertices.clear();
    unsigned int nnode=0;
    for (size_t k=0; k<_edges.size(); ++k)
    {
        for (size_t j=0; j<_edges[k].size(); ++j)
        {
            nnode=max(nnode,_edges[k][j]);
            _vertices.insert(_edges[k][j]);       // graph_2
        }
    }
    if (_edges.size()>0) _maxvert=nnode;          // more than 1 edge i graph?
 }

ostream& operator<<(ostream &s, graph const &rhs)
{
    s << "Graph with  " << rhs.Nedges() << " edges and  " << rhs.Nvertices() << " vertices" << endl;

    auto &edges=rhs._edges;
    s << "\n --  Edges  --\n";
    for (size_t k=0; k<edges.size(); ++k)
    {
        s << k << " : ";
        for (size_t j=0; j<edges[k].size(); ++j)
        {
            s << edges[k][j] << "  ";
        }
        s << endl;
    }

    s << "\n --  Vertices  --\n";                   // graph_2
    for (auto v: rhs._vertices)                     // graph_2
    {
        s << v << "  ";
    }
    s << endl;

    return s;
```

```
}

```

```
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

#include <array>
#include <iostream>
#include <set>                // graph_2
#include <string>
#include <vector>

/**
  A better graph class that doesn't requires a consecutive numbering of the vertices.
*/
class graph {
public:
    /** \brief Reads edges for graph from file.
     *
     *  If the file  @p file_name does not exist then the code stops with an appropriate message.
     *
     *  A consecutive numbering of the vertices is required.
     *
     * @param[in]    file_name   name of the ASCII-file
     */
    graph(const std::string &file_name);

    graph(graph const & org) = default;
    graph& operator=(graph const & rhs) = default;

    /**
      Determines the neighboring vertices for each node from the edge definition.
      The node itself is not contained in the neighboring vertices.

      @return       vector[nn][*] with all neighboring vertices for each node
    */
    std::vector<std::vector<unsigned int>> get_node2nodes() const;

    /**
      @return       number of edges
    */
    size_t Nedges() const
    {
        return _edges.size();
    }

    /**
      @return       number of vertices
    */
    size_t Nvertices() const
    {
        return _vertices.size();                  // graph_2
    }

     /**
      @return       largest vertex index
      */
    size_t Max_vertex() const             // graph_2
    {
        return _maxvert;
    }

    friend std::ostream& operator<<(std::ostream &s, graph const &rhs);

private:
    /**
      Determines the number of vertices from the edge information.
      No consecutive numbering of the vertices required.
    */
    void DetermineNumberVertices();

    std::vector<std::array<unsigned int, 2>> _edges;   /**< stores the two vertices for each edge */
    std::set<unsigned int>                   _vertices;/**< stores the vertex indices */
//    size_t                                   _nvert;   /**< number of vertices */
    int                                      _maxvert; /**< maximal vertex index */

};


#endif // GRAPH_H_INCLUDED
```

```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_2.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;
}
```