```
    1: #include "graph.h"
    2: #include <algorithm>
    3: #include <array>
    4: #include <cassert>
    5: #include <fstream>
    6: #include <iostream>
    7: #include <map>
    8: #include <stdexcept>
    9: #include <string>
   10: #include <vector>
   11: using namespace std;
   12:
   13: void read_edges_from_file(const string &file_name, vector<array<int,
2>> &v)
   14: {
   15:     ifstream fin(file_name);            // Oeffne das File im ASCII-Mo
dus
   16:     if ( fin.is_open() ) {              // File gefunden:
   17:         v.clear();                      // Vektor leeren
   18:         int k,l;
   19:         while ( fin >> k >> l) {v.push_back({k,l});} // Einlesen
   20:         if (!fin.eof()) {
   21:             // Fehlerbehandlung
   22:             cout << " Error handling \n";
   23:             if ( fin.bad() )  {throw runtime_error("Schwerer Fehler i
n istr");}
   24:             if ( fin.fail() ) { // Versuch des Aufraeumens
   25:                 cout << " Failed in reading all data.\n";
   26:                 fin.clear();
   27:             }
   28:         }
   29:         v.shrink_to_fit();
   30:     }
   31:     else {                              // File nicht gefunden:
   32:         cout << "\nFile " << file_name << " has not been found.\n\n"
;
   33:         assert( fin.is_open() && "File not found." );       // exepti
on handling for the poor programmer
   34:     }
   35:     //return;
   36: }
   37:
   38:
   39: map<int, vector<int>> get_node2nodes(vector<array<int,2>> const & edg
es)
   40: {
   41:     // We allow a non-continuous numbering of nodes.
   42:     // Determine the neighborhood for each vertex.
   43:     map<int, vector<int>> n2n;
   44:     for (size_t k=0; k<edges.size(); ++k)
   45:     {
   46:         const int v0 = edges[k][0];
   47:         const int v1 = edges[k][1];
   48:         n2n[v0].push_back(v1);          // add v1 to neighborhood of v
0
   49:         n2n[v1].push_back(v0);          //     and vice versa
   50:     }
```

```
51:     // ascending sort of entries per node
52:     for (auto [key, value]: n2n)
53:     {
54:         sort(value.begin(),value.end());
55:     }
56:
57:     return n2n;
58: }
59:
60:
```

```
    1: #pragma once                  // substitutes header guarding
    2:
    3: #include <array>
    4: #include <map>
    5: #include <string>
    6: #include <vector>
    7:
    8: /**
    9:   This function opens the ASCII-file named @p file_name and reads the
   10:   int data into the C++ vector @p v.
   11:   If the file  @p file_name does not exist then the code stops with a
n appropriate message.
   12:   @param[in]    file_name   name of the ASCII-file
   13:   @param[out]   v           C++ vector with edge vertices
   14: */
   15:
   16: void read_edges_from_file(const std::string& file_name, std::vector<s
td::array<int,2>>& v);
   17:
   18:
   19: /**
   20:   Determines the neighboring vertices for each node from the edge def
inition @p edges .
   21:   The node itself is not contained in the neighboring vertices.
   22:
   23:   @param[in]    edges   vector[ne][2] with edge vertices
   24:   @return       vector[nn][*] with all neighboring vertices for each
node
   25: */
   26: std::map<int, std::vector<int>> get_node2nodes(std::vector<std::array
<int,2>> const & edges);
   27:
   28:
```

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

int main()
{
    cout << "Hello Graph!" << endl;
    const string name{"g_1_map.txt"};

    // read the edges
    vector<array<int,2>> edges;
    read_edges_from_file(name, edges);

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

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

    cout << "\n --  Nodes to Node  --\n";
    for (auto const & [key, value]: n2n)
    {
        cout << key << " : ";                    // node number
        for (size_t j=0; j<value.size(); ++j)  //   its neighborhood
        {
            cout << value.at(j) << " ";
        }
        cout << endl;
    }

    return 0;
}
```