```
//***************************************************************************************/
// DS2 Exercise 04: Directed Graph class by Wu, Yi-Hung@ICE.CYCU
//***************************************************************************************/
#include <iostream>                                     // cout, endl
#include <fstream>                                      // open, is_open, close, ignore
#include <string>                                       // string, find_last_of, substr
#include <vector>                                       // vector, push_back
#include <cstdlib>                                      // system, atoi
#include <cstring>                                      // strcpy
#include <iomanip>                                      // setw
#include <queue>                                        // queue: push, pop, front, empty
#include <stack>                                        // stack: push, pop, top, empty
#include <algorithm>                                    // sort

using namespace std;
#define MAX_LEN     10                                  // array size of student id and name
#define PAGE        25                                  // amount of display on screen
#define NONE        -1                                  // error flag

class DirectedGraph
{
     typedef struct sP                                  // student pair
   {    char      sid1[MAX_LEN];                        // 1st sid: sender
        char      sid2[MAX_LEN];                        // 2nd sid: receiver
        float     wgt;                                  // pair weight
   }    studentPair;

   typedef struct aLN                                   // node of adjacency lists
   {    string         sid2;                            // receiver
        float          weight;                          // pair weight
        struct aLN     *next;                           // pointer to the next node
   }    adjListNode;

   typedef struct aL                                    // adjacency list
   {    string         sid1;                            // sender
        adjListNode *head;                              // pointer to the first node of a list
        int            inf;                             // influence value
   }    adjList;
```

```
    vector<adjList> adjL;                               // the adjacency lists
    string          fileNO;                             // a number to form a file name
    float           wgtLB;                              // lower bound of weights
    //*********************************************************************/
    //  the above are private data members
    //*********************************************************************/

    bool readF(vector<studentPair> &);                  // get all records from a file
    void insert(adjList &);                             // insert an adjacency list
    int locate(vector<adjList> &, string &);            // locate the index in adjacency lists
    int locate(string &key)   { return locate(adjL, key); }  // locate the index in adjacency lists
    bool addCount(adjListNode *, adjListNode *);        // count only if not visited yet
    void saveINF(vector<adjList> &, string);            // write influence values as a file
    void clearUp(vector<adjList> &);                    // release the space of adjacency lists
    //*********************************************************************/
    //  the above are private methods
    //*********************************************************************/

public:
    DirectedGraph(): fileNO(""), wgtLB(0) {}            // default constructor
    DirectedGraph(DirectedGraph &obj): adjL(obj.adjL), fileNO(obj.fileNO), wgtLB(0)
    {    }                                              // shallow copy constructor
    bool existed() {      return adjL.size(); }         // check the existence
    void setLB(float v) {     wgtLB = v;    }           // set up the value of wgtLB

    bool create();                                      // read pairs from a file into adjacency lists
    void saveF();                                       // write adjacency lists as a file
    void compINF(string);                               // compute influence values by BFS
    void compINF();                                     // compute influence values by DFS

    void clearUp()    {     clearUp(adjL);    }         // destroy the object
    ~DirectedGraph()  {     clearUp();    }             // destructor: destroy the object
};   // class DirectedGraph
```