#include <cs50.h>

#include <stdio.h>

#include <string.h>

// Max number of candidates

#define MAX 9

// preferences[i][j] is number of voters who prefer i over j

int preferences[MAX][MAX];

// locked[i][j] means i is locked in over j

bool locked[MAX][MAX];

// Each pair has a winner, loser

typedef struct

{

int winner;

int loser;

}

pair;

// Array of candidates

string candidates[MAX];

pair pairs[MAX \* (MAX - 1) / 2]; //pairs为每个两两竞赛，n（n-1）/2为一共要进行多少场比赛。

int pair\_count;

int candidate\_count;

// Function prototypes

bool vote(int rank, string name, int ranks[]);

void record\_preferences(int ranks[]);

void add\_pairs(void);

void sort\_pairs(void);

void lock\_pairs(void);

void print\_winner(void);

bool has\_cycle(int winner, int loser);

int main(int argc, string argv[])

{

// Check for invalid usage

if (argc < 2)

{

printf("Usage: tideman [candidate ...]\n");

return 1;

}

// Populate array of candidates

candidate\_count = argc - 1;

if (candidate\_count > MAX)

{

printf("Maximum number of candidates is %i\n", MAX);

return 2;

}

//candidates是命令行参数，从一开始就传进去了

for (int i = 0; i < candidate\_count; i++)

{

candidates[i] = argv[i + 1];

}

// Clear graph of locked in pairs

for (int i = 0; i < candidate\_count; i++)

{

for (int j = 0; j < candidate\_count; j++)

{

locked[i][j] = false;

}

}

pair\_count = 0;

int voter\_count = get\_int("Number of voters: ");

// Query for votes

for (int i = 0; i < voter\_count; i++)

{

// ranks[i] is voter's ith preference

int ranks[candidate\_count];

// Query for each rank

for (int j = 0; j < candidate\_count; j++)

{

string name = get\_string("Rank %i: ", j + 1);

if (!vote(j, name, ranks))

{

printf("Invalid vote.\n");

return 3;

}

}

record\_preferences(ranks);

printf("\n");

}

add\_pairs();

sort\_pairs();

lock\_pairs();

print\_winner();

return 0;

}

//因为要用编号代替names对candidates（候选人）进行排序，所以这里创造一个函数来赋予候选人index（索引）。

//遍历命令行参数，例如a b c。当voter输入b，在candidates[1]的位置找到了b，于是把b的名字换成了candidate\_index == 1。

int get\_index(string name)

{

for (int i = 0; i < candidate\_count; i++)

{

if (strcmp(candidates[i], name) == 0)

return i;

}

return -1;

}

// Update ranks given a new vote

bool vote(int rank, string name, int ranks[])

{

// TODO

int candidate\_index = get\_index(name);

if (candidate\_index != -1)

{

ranks[rank] = candidate\_index; //ranks[rank] rank是给定的！用户在rank1 2 3里面打字，最终转为candidates的索引。

//那么，ranks这个数组最终就会按voter的偏好顺序存储candidates的索引

return true;

}

return false;

}

// Update preferences given one voter's ranks

void record\_preferences(int ranks[])

//传入的是一个排名。例如， ranks[]= 2 0 1（这里的2 0 1只是代码，代表在命令行第几个打进candidate的名字）

//这是一个i行j列的二维数据结构。遍历每个voter的输入的时候，一旦先检测到i行的candidate，

//（说得更具体些，是他的索引），那么就把排在[i][j]所在这一行右边的所在元素的preferences[i][j]（意思是i胜过j）进行+1.

// a b c

//a. 0 1 2 这一行表达 有一个voter prefer a候选人 than b候选人，有两个voter prefer A than C.

//b. 1 0 3

//c. 2 3 0

{

for (int i = 0; i < candidate\_count; i++)

{

for (int j = i + 1; j < candidate\_count; j++)

preferences[ ranks[i] ][ ranks[j] ]++; //这个++就是在具体i胜过j（被voter prefer i than j）多少次上面加1（来扩大胜利的强度）

}

return;

}

// Record pairs of candidates where one is preferred over the other

void add\_pairs(void)

{

for (int i = 0; i < candidate\_count; i++)

{

for (int j = 0; j < candidate\_count; j++)

{

if (preferences[i][j] > preferences[j][i]) //如果ij竞争中i胜利

{

pair new\_pair = {i, j};

pairs[pair\_count] = new\_pair; //populate pairs这个数组。（里面存储着每个两两比赛中谁赢谁输）

pair\_count++;

}

}

}

}

int pair\_weight(i)

{

int pair\_winner = pairs[i].winner;

int pair\_loser = pairs[i].loser;

return preferences[pair\_winner][pair\_loser]; //完全看不懂这里

}

// Sort pairs in decreasing order by strength of victory

void sort\_pairs(void)

{

//use reversed selection sort.

for (int i = pair\_count - 1; i > 0; i--) //例如有9对，那就从索引8开始

{

int min\_weight = pair\_weight(i);

int min\_idx = i;

for (int j = i - 1; j >= 0; j--) //从索引8 往索引7进行第一次对比

{

if (pair\_weight(j) < min\_weight)

{

min\_weight = pair\_weight(j);

min\_idx = j; //刷新最小权重和最小权重对应的索引

}

}

}

//swap strongest win with current index

pair temp = pairs[min\_idx];

pairs[min\_idx] = pairs[i];

pairs[i] = temp;

}

bool has\_cycle(int winner, int loser)

{

if (locked[loser][winner] == true) //如果已经有loser指向winner了就直接真。

{

return true;

}

for (int i = 0; i < candidate\_count; i++)//

{

if(locked[loser][i] == true && has\_cycle(winner, i)) //如果loser经过i，并且winner和i之间也有循环（递归调用），则假

{

return true;

}

}

return false; //如果经过了所有测试，就返回false

}

// Lock pairs into the candidate graph in order, without creating cycles

void lock\_pairs(void)

{

for(int i = 0; i < pair\_count; i++)

{

int winner = pairs[i].winner; //为什么这里是两个int啊

int loser = pairs[i].loser;

if(!has\_cycle(winner, loser))

{

locked[winner][loser] = true;

}

}

}

// Print the winner of the election

// a b c

//a. 0 1 2

//b. 1 0 3

//c. 2 3 0

void print\_winner(void)

{

for(int row = 0; row < candidate\_count; row++)

{

for(int col = 0; col < candidate\_count; col++)

{

if(locked[row][col] == true) //例如locked[1][2]就是a锁定了b

{

break;//继续下一列遍历

}

else if(col == candidate - 1)//当col遍历完成

{

printf("%s", candidates[row]); //？？为什么现在就可以print赢家？？

}

}

}

return;

}