```c
#include <cs50.h>
#include <stdio.h>
#include <string.h>

// Max número de candidatos.
#define MAX 9

// preferences[i][j], array de preferencia de votos dos eleitores.
int preferences[MAX][MAX];

// locked[i][j] trava o i em comparação com o j.
bool locked[MAX][MAX];

// Cada par tem um vencedor e um perdedor.
typedef struct
{
    int winner;
    int loser;
}
pair;

// Array de candidatos.
string candidates[MAX];
pair pairs[MAX * (MAX - 1) / 2];

int pair_count;
int candidate_count;

// Protótipos de funções.
bool vote(int rank, string name, int ranks[]);
void record_preferences(int ranks[]);
void add_pairs(void);
void sort_pairs(void);
bool has_cycle(int winner, int loser);
void lock_pairs(void);
void print_winner(void);

int main(int argc, string argv[])
{
    // Verifica usos inválidos.
    if (argc < 2)
    {
        printf("Usage: tideman [candidate ...]\n");
        return 1;
    }

    // Preenche o array de candidatos.
    candidate_count = argc - 1;
    if (candidate_count > MAX)
```

```
{
    printf("Maximum number of candidates is %i\n", MAX);
    return 2;
}
for (int i = 0; i < candidate_count; i++)
{
    candidates[i] = argv[i + 1];
}

// Limpa o gráfico de pares travados.
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: ");

// Solicita os votos.
for (int i = 0; i < voter_count; i++)
{
    // ranks[i] é a preferência de voto de cada eleitor [i].
    int ranks[candidate_count];

    // Solicita cada 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;
```

```
}

// Armazena o voto dentro do array ranks[rank].
// rank = nº do voto; ranks[candidato naquela preferência].
bool vote(int rank, string name, int ranks[])
{
    for(int i = 0; i < candidate_count; i++)
    {
        if(strcmp(candidates[i], name) == 0)
        {
            ranks[rank] = i;
            return true;
        }
    }
    return false;
}

// Armazena as preferências de votos.
// Ex: 1º voto = Alice e 2º voto = Bob então 0,1 = 1;
// Ex2: 2º voto = Alice e 3º voto = Bob então 0,1 = 1 + 1 = 2.
void record_preferences(int ranks[])
{
    for (int linha = 0; linha < candidate_count; linha++)
    {
        for(int coluna = linha + 1; coluna < candidate_count; coluna++)
        {
            preferences[ranks[linha]][ranks[coluna]]++;
        }
    }
    return;
}

// Grava os pares de candidatos onde um é favorito em comparação ao
outro.
void add_pairs(void)
{
    for(int linha = 0; linha < candidate_count; linha++)
    {
        for(int coluna = 0; coluna < candidate_count; coluna++)
        {
            if(preferences[linha][coluna] > preferences[coluna][linha])
            {
                pairs[pair_count].winner = linha;
                pairs[pair_count].loser = coluna;
                pair_count++;
            }
        }
    }
    return;
```

```
}

// Organiza os pares em ordem decrescente.
void sort_pairs(void)
{
    int posição, força[2] = {0, 0};

    for(int linha = 0; linha < pair_count - 1; linha++)
    {
        posição = linha; // salva a posição do valor mais alto
        // Salva a diferença de votos do par. Ex: posição 0,1 = 5,
posição 1,0 = 1, força[0] = 5 - 1 = 4.
        força[0] = preferences[pairs[linha].winner][pairs[linha].loser] -
preferences[pairs[linha].loser][pairs[linha].winner];

        for(int coluna = linha + 1; coluna < pair_count; coluna++)
        {
            // Salva a diferença de votos do próximo par.
            força[1] =
preferences[pairs[coluna].winner][pairs[coluna].loser] -
preferences[pairs[coluna].loser][pairs[coluna].winner];

            // Caso a força do próximo par seja maior que a do primeiro,
salva a posição do mesmo.
            if(força[1] > força[0])
            {
                posição = coluna;
            }
        }

        // Troca a posição dos pares.
        pair backup = pairs[linha];
        pairs[linha] = pairs[posição];
        pairs[posição] = backup;
    }
    return;
}

// Verifica se tem um ciclo nos vencedores.
bool has_cycle(int winner, int loser)
{
    // Se já existe uma "seta" no sentido contrário, não se pode travar
essa.
    if(locked[loser][winner] == true)
    {
        return true;
    }
```

```
    // Verifica se já existe uma "seta" apontada para os outros
candidatos.
    for(int i = 0; i < candidate_count; i++)
    {
        // Se já existe uma "seta" de um candidato(i) apontada para o
perdedor, refaz o ciclo para ver se já existe uma seta contrária entre o
vencedor e esse candidato(i)
        if(locked[loser][i] == true && has_cycle(winner, i))
        {
            // Se houver, retorna true, impedindo essa "seta" de ser
travada e gerar um ciclo.
            return true;
        }
    }

    // Retorna falso se a "seta" puder ser criada.
    return false;
}

// Trava os pares, "criando setas".
void lock_pairs(void)
{
    // Roda entre todos os pares.
    for(int i = 0; i < pair_count; i++)
    {
        // salva o número referente a cada candidato.
        int winner = pairs[i].winner;
        int loser = pairs[i].loser;

        // Roda o has_cycle para ver se a "criação dessa seta" gera um
ciclo ou não.
        if(!has_cycle(winner, loser))
        {
            locked[winner][loser] = true;
        }
    }
    return;
}

// Imprime o vencedor da eleição.
void print_winner(void)
{
    // verifica todos os candidatos.
    for(int linha = 0; linha < candidate_count; linha++)
    {
        // Variável para ver se existe uma seta apontada ao candidato.
        bool winner = false;

        // Verifica os d+ candidatos em realação ao primeiro.
```

```c
        for(int coluna = 0; coluna < candidate_count; coluna++)
        {
            // Verifica se existe uma seta apontada ao candidato, caso
existe o wiiner se torna verdadeiro e passa ao próximo candidato.
            if(locked[coluna][linha])
            {
                winner = true;
                break;
            }
        }

        // Se winner continua em false após iteragir com todos os
candidatos, significa que o candidato não tem uma seta apontada para ele,
ou seja, é o vencedor.
        if(!winner)
        {
            printf("%s\n", candidates[linha]);
        }
    }
    return;
}
```