Report

1. Describe how a bingo matrix is generated by hidden parameters? What are hidden parameters?

   Answer: hidden parameters include two values, “jump” and “shift” in Bingo.cpp file.

   The matrix is filled with the number **1** to **the size**, supposed **the size** equals to 25, which means the matrix spans **1** to **25**.

   The “shift” hidden parameter means by which index to divide the matrix into two parts, first part is from index 0 to index “shift - 1”, while the second is from “shift” to “the size - 1”. For example, if the size equals to 25 and “shift” is 5, the first part is from mat[0]-mat[4], and the second part is mat[5]-mat[24] with mat[5] = 1.

   While the hidden parameter “jump” means how many differences between the next one and the current one, and if the next one is more than the size, its value should module the “jump” and plus 1. From example, given the size is 25, “shift” is 5 and “jump” is 8, then if the current one is 9, the next one will be 9+jump, it is 17, and if the current one is 25, the next one will be (25+jump)%jump + 1, it is 2.

2. Explain the strategy of Junhee and Colin in the given program

   Answer:

   Junhee: pick the unmarked one to mark in order from index 0 to the last one.

   Colin: from the unmarked group randomly pick one to mark.

3. Build a strategy that does not guess the opponent matrix

   Answer:

   Strategy:

   (1) when it just begins the game, could choose any one, suppose 1

   (2) find the row (r) that has the least unmarked numbers except 0, and count how many unmarked numbers (rn)

   (3) find the column (c) that has the least unmarked numbers except 0, and count how many unmarked numbers (cn)

   (4) find the diagonal that has the least unmarked numbers (dn) except 0, the diagonal from (0, 0) to (size-1, size-1) is expressed as 0, while the diagonal from (0, size-1) to (size-1, 0) is expressed as 1

   (5) compare rn, cn, and dn, if rn is the smallest, choose unmarked one from that row (r), while if cn is the smallest, choose unmarked one from that column (c), otherwise choose unmarked one from that diagonal.

   (6) repeat (5) until Bingo is reached.

   Code:

```
/*********************************************************
 This is the code you need to modify.

 IMPORTANT NOTE
 1. Change "2016000000" in the filename, class name,
```

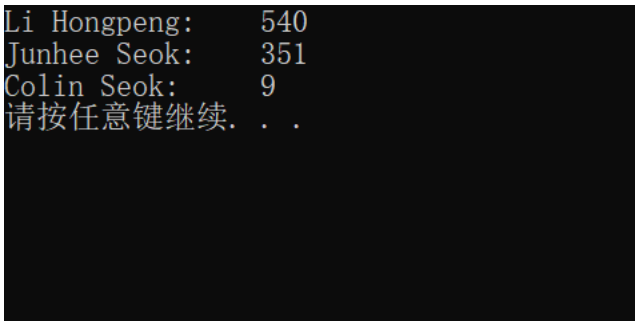
```
   preprocessing statements to your real student id.
 2. Change "MyName" in the constructor to your real name.

 These two changes are very important to grade your submission.
 If you miss the changes, you will have a penalty.

**********************************************************/

// change "2016000000" to your real student id
#ifndef _BINGO_2016171097_H_
#define _BINGO_2016171097_H_

#include "setting.h"
#include "Bingo.h"

// change "2016000000" to your real student id
class Bingo_2016171097 : public Bingo
{
public:
    // change "2016000000" to your real student id
    Bingo_2016171097() {
        mName = "Li Hongpeng";    // change "MyName" to your real name in English
    }
    // change "2016000000" to your real student id
    ~Bingo_2016171097() { /* add whatever */ }

    int myCall(int *itscalls, int *mycalls, int ncalls);

private:
    // declare whatever you want
    int findShift() {
        int shift = -1;
        for (int i = 0; i < mSize*mSize; i++) {
            if (mData[i] == 1) {
                shift = i;
                break;
            }
        }
        return shift;
    }

    int findJump() {
        int jump = -1;
        int i = 0;
```

```
        while (jump == -1) {
            if (mData[i + 1] > mData[i]) {
                jump = mData[i + 1] - mData[i];
            }
        }
        return jump;
    }
};


/*********************************************************
  itcalls: a vector of int variables with size of "ncalls"
           showing the opponent's calls for the previous turns
  mycalls: a vector of int variables with size of "ncalls"
           showing my calls for the previous turns
  ncalls: number of previous turns
*********************************************************/
int Bingo_2016171097::myCall(int *itscalls, int *mycalls, int ncalls)
{
    // implement your strategy

    //when it just begins the game, could choose any one, suppose 1
    if (ncalls == 0) {
        return 1;
    }

    //find the row that has the least unmarked numbers except 0
    //count how many unmarked numbers, rn
    int rn = mSize;
    int r = 0;
    for (int i = 0; i < mSize; i++) {
        int tn = 0;
        for (int j = 0; j < mSize; j++) {
            if (mMark[mat2vec(i, j)] == false) {
                tn++;
            }
        }
        if (tn != 0 && rn > tn) {
            rn = tn;
            r = i;
        }
    }

    //find the column that has the least unmarked numbers except 0
```

```
//count how many unmarked numbers, cn
int cn = mSize;
int c = 0;
for (int i = 0; i < mSize; i++) {
    int tn = 0;
    for (int j = 0; j < mSize; j++) {
        if (mMark[mat2vec(j, i)] == false) {
            tn++;
        }
    }
    if (tn != 0 && cn > tn) {
        cn = tn;
        c = i;
    }
}

//find the diagonal that has the least unmarked numbers except 0
//count how many unmarked numbers, dn1, dn2
int dn1 = 0;
int d1 = 0;
for (int i = 0; i < mSize; i++) {
    if (mMark[mat2vec(i, i)] == false) {
        dn1++;
    }
}
if (dn1 == 0) {
    d1 = mSize;
}

int dn2 = 0;
int d2 = 1;
for (int i = 0; i < mSize; i++) {
    if (mMark[mat2vec(i, mSize - i - 1)] == false) {
        dn2++;
    }
}
if (dn2 == 0) {
    d2 = mSize;
}

//compare rn, cn, dn1, dn2, find the smallest one and pick one from it
if (rn <= cn && rn <= dn1 && rn <= dn2) {
    //choose from row
    for (int j = 0; j < mSize; j++) {
```

```
            if (mMark[mat2vec(r, j)] == false) {
                return mData[mat2vec(r, j)];
            }
        }
    }
    else if (cn <= rn && cn <= dn1 && cn <= dn2) {
        //choose from column
        for (int j = 0; j < mSize; j++) {
            if (mMark[mat2vec(j, c)] == false) {
                return mData[mat2vec(j, c)];
            }
        }
    }
    else if (dn1 <= rn && dn1 <= cn && dn1 <= dn2) {
        //choose from diagonal
        for (int i = 0; i < mSize; i++) {
            if (mMark[mat2vec(i, i)] == false) {
                return mData[mat2vec(i, i)];
            }
        }
    }
    else {
        //choose from diagonal
        for (int i = 0; i < mSize; i++) {
            if (mMark[mat2vec(i, mSize - i - 1)] == false) {
                return mData[mat2vec(i, mSize - i - 1)];
            }
        }
    }

    return 1;
}

#endif
```

Running:



4. Build a strategy that does guess the opponent matrix

5. Build a strategy that guesses the opponent matrix as well as tries to confuse the opponent's guess assuming your opponent also tries to guess

6. Among the strategies of problem 3~5, pick one to submit for the full league with your classmates
   Answer: choose the code written in problem 3