为防止不同编译器中文乱码的情况，以下附上完整代码。

main.cpp函数

#include "print.h"

// 其他库函数、命名空间等在print.h头文件中已定义

bool play(chessboard &board, int &num, char &ch) // 执行游戏函数

{

    int x, y;

    bool result;

    do

    {

        if (num % 2 == 0)

            cout << endl

                 << "玩家1执棋." << endl;

        else

            cout << endl

                 << "玩家2执棋." << endl;

        print\_display(board);               // 是否打印棋盘的函数

        result = print\_retract(board, num); // 是否悔棋或者退出棋局的函数

        if (!result)

            return false;           // 只有中途输入q会退出棋局

        print\_tip(board, num);      // 是否获取建议的函数

        print\_gochess(board, x, y); // 确定落子索引的函数

        result = print\_judge(num, board, x, y);

        // 下棋并判断该局是否结束的函数，返回值为该步后比赛结果：false有两种情况：一是225个格子下满，二是该步后未分出胜负

    } while (!result && num < 225); // 循环，result是错的并且num<225时候能执行

    ch = print\_final(board, result, num);

    return true; // 对于整个函数来讲，只会返回true，与中途退出的false（玩家主动结束游戏）相对应

}

int main()

{

    // 一些初始化内容

    const int n = 15;    // 表示棋盘大小

    chessboard board(n); // 初始化棋盘

    int num = 0;         // num实时记录棋盘上的棋子个数

    char ch;             // 用来代表每次输入的命令

    bool result;         // 用于一些需要判断对错的情况

    ofstream outfile;    // 用于保存记录

    ifstream infile;     // 用于读取记录

    // 自动输出游戏说明

    cout << '\t' << "---欢乐五子棋---" << endl

         << endl;

    cout << "游戏说明" << endl;

    cout << "\t这是一个简易五子棋游戏\n"

            "在每一回合中,输入0~14之间的正整数数对表示落子索引\n"

            "输入r可以回到当前玩家所在的上一回合(注意: 每次重开应用后,需要至少下两步棋才允许悔棋)\n"

            "输入t可以获得当前棋局下的建议落子位置(仅作为参考)\n"

            "输入q可以退出棋局并保存\n"

            "如果棋局胜负已分,您可以选择悔棋回到上一步,您也可以选择保存并退出程序,但是并不支持读取记录后悔棋(仅该步不支持)\n"

         << endl;

    // 是否读取上次记录

    infile.open("save\_file");

    if (!infile)

    {

        cout << "输入s开始游戏" << endl;

        cin >> ch;

        while (ch != 's')

        {

            cout << endl

                 << "请正确输入指令." << endl;

            cin >> ch;

        }

    }

    else

    {

        cout << "输入s开始新游戏,输入i读取上一次游戏存档。" << endl;

        cin >> ch;

        while (ch != 's' && ch != 'i')

        {

            cout << endl

                 << "请正确输入指令." << endl;

            cin >> ch;

        }

        if (ch == 's')

            infile.close();

        else

            board.in(infile, n, num);

    }

    cout << "-------------------------------------------------------" << endl;

    // 开始进行游戏

AGAIN:                             // goto标志，如果分出胜负后悔棋并继续游戏，回到该步

    result = play(board, num, ch); // 游戏函数，游戏中玩家输入q会退出，函数返回false 其他情况返回true

    if (!result) // 中途输入q退出的时候

    {

        outfile.open("save\_file");   // 打开文件

        board.save(outfile, n, num); // 保存记录的函数

        cout << "该局记录已保存." << endl;

        cout << "输入回车结束程序..." << endl;

        cin.get(ch);

        cin.get(ch);

        return 0; // 程序结束

    }

    // result是true，棋盘下满或者胜负已分

    switch (ch)

    {

    case 'r':

        board.retract(); // 悔棋

        num = num - 2;

        goto AGAIN; // 继续游戏

    case 'q':

        outfile.open("save\_file");   // 打开文件

        board.save(outfile, n, num); // 保存记录的函数

        cout << "该局记录已保存." << endl;

        cout << "输入回车结束程序..." << endl;

        cin.get(ch);

        cin.get(ch);

        return 0; // 程序结束

    default:

        cout << "输入回车结束程序..." << endl;

        cin.get(ch);

        cin.get(ch);

    }

    return 0;

}

print.h头文件

// print.h头文件，是游戏主要过程的分步拆解

#ifndef PRINT\_H

#define PRINT\_H

#include "chess.h"

// 分解落子索引

void print\_gochess(chessboard &board, int &x, int &y);

// 分解下棋

bool print\_judge(int &num, chessboard &board, int &x, int &y);

// 分解打印棋盘格

void print\_display(chessboard &board);

// 分解悔棋或者退出

bool print\_retract(chessboard &board, int &num);

// 分解给出下棋建议

void print\_tip(chessboard &board, const int &num);

// 分出胜负后，用户给出下一步操作

char print\_final(chessboard &board, bool result, const int &num);

#endif

print.cpp文件

#include "print.h"

void print\_gochess(chessboard &board, int &x, int &y)

{

    /\*\*

     \* @brief  用户输入落子索引，判断是否合法，是否确定输入

     \* @param  x 落子横坐标

     \* @param  y 落子纵坐标

     \* @param  ch 用户输入命令

     \* @param  result 判断落子合法性

     \*/

    char ch;

    bool result;

    do

    {

        do

        {

            cout << "请输入落子索引(第一个整数为横轴坐标,第二个数为纵轴坐标)" << endl;

            cin >> x >> y;

            result = board.avail(x, y); // 判断合法性

        } while (!result);

        do

        {

            cout << "是否确认落子?确认请输入y,反之请输入n." << endl;

            cin >> ch;

        } while (ch != 'y' && ch != 'n');

    } while (ch != 'y');

}

bool print\_judge(int &num, chessboard &board, int &x, int &y)

{

    /\*\*

     \* @brief  判断棋局是否结束

     \* @param  x 落子横坐标

     \* @param  y 落子纵坐标

     \* @param  num 棋盘上棋子个数

     \* @param  result 判断棋局是否结束

     \* @return true表示有玩家获胜，false表示胜负未分或者棋盘已满

     \*/

    bool result;

    if (num % 2 == 0)

    {

        board.go\_chess(x, y, '%');       // 玩家一

        result = board.judge(x, y, '%'); // 判断棋局

    }

    if (num % 2 == 1)

    {

        board.go\_chess(x, y, '#'); // 玩家二

        result = board.judge(x, y, '#');

    }

    ++num;

    if (num == 225)

        return false;

    return result;

}

void print\_display(chessboard &board)

{

    /\*\*

     \* @brief  用户选择是否打印棋盘格

     \* @param  ch 用户输入命令

     \*/

    char ch;

    do

    {

        cout << "若想打印棋盘格,请输入p,反之请输入n." << endl;

        cin >> ch;

        if (ch == 'p')

            board.display(); // 打印棋盘格

    } while (ch != 'p' && ch != 'n');

}

bool print\_retract(chessboard &board, int &num)

{

    /\*\*

     \* @brief  用户选择是否悔棋或退出棋局

     \* @param  num 棋盘上棋子个数

     \* @return false表示退出棋局

     \*/

    char ch;

    do

    {

        cout << "若需要悔棋,请输入r,若退出游戏,请输入q,若均不,请输入n."

             << endl;

        cin >> ch;

        if (ch == 'r')

        {

            if (board.record\_num() <= 2)

                cout << "悔棋失败." << endl;

            else

            {

                board.retract();

                num = num - 2;

                cout << "悔棋后,棋盘如下所示." << endl;

                board.display();

            }

        }

        if (ch == 'q')

            return false;

    } while (ch != 'r' && ch != 'n' && ch != 'q');

    return true;

}

void print\_tip(chessboard &board, const int &num)

{

    /\*\*

     \* @brief  用户选择是否需要落子建议

     \* @param  ch 用户输入命令

     \* @param  num 棋盘上棋子个数

     \*/

    char ch;

    do

    {

        cout << "是否需要建议,若需要,请输入t,反之,请输入n."

             << endl;

        cin >> ch;

        if (ch == 't')

        {

            if (num % 2 == 0)

                board.evaluate('%'); //

            else

                board.evaluate('#');

        }

    } while (ch != 't' && ch != 'n');

}

// 分出胜负后，用户给出下一步操作

char print\_final(chessboard &board, bool result, const int &num)

{

    char ch;

    board.display();

    if (result)

    {

        if (num % 2 == 1)

            cout << "玩家1获胜!" << endl;

        else

            cout << "玩家2获胜!" << endl;

    }

    else

        cout << "棋盘已满,胜负未分,该局平局." << endl;

    cout << endl

         << "悔棋请输入r,输入q保存记录并退出游戏,输入其他退出游戏." << endl;

    cin >> ch;

    return ch;

}

chess.h头文件

// chessnode和chessboard的头文件

#ifndef CHESS\_H

#define CHESS\_H

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

class chessnode // 棋子类

{

private:

    int location\_x; // 横坐标

    int location\_y; // 纵坐标

    char signal;    //  玩家一棋子%    玩家二棋子#

    int value1, value2;

    //  评估值，即在该点落子后能连成的长度

    //  value1 %  value2 #

public:

    chessnode() // 构造函数

    {

        signal = 'o';

        value1 = 0;

        value2 = 0;

    }

    void set\_chessnode(int x, int y) // 设置横纵坐标

    {

        location\_x = x;

        location\_y = y;

    }

    int get\_x() const // 获取横坐标

    {

        return location\_x;

    }

    int get\_y() const // 获取纵坐标

    {

        return location\_y;

    }

    void set\_signal(char ch) // 设置棋子状况

    {

        signal = ch;

    }

    char get\_signal() const // 返回棋子状况

    {

        return signal;

    }

    void set\_value1(int x) // 设置value1

    {

        value1 = x;

    }

    int get\_value1() const // 返回value1

    {

        return value1;

    }

    void set\_value2(int x) // 设置value2

    {

        value2 = x;

    }

    int get\_value2() const // 返回value2

    {

        return value2;

    }

};

class chessboard // 棋盘类

{

private:

    chessnode \*\*board;                // 二维指针，指向棋盘

    int size;                         // 棋盘大小

    vector<chessnode> record;         // 记录下棋顺序

    vector<chessnode> value1;         // 记录player1能连成长度大于等于3的棋子

    vector<chessnode> value2;         // 记录player2能连成长度大于等于3的棋子

    int value(int x, int y, char id); // 判断棋盘(x,y)处落子id后能连成的最大长度

public:

    chessboard(int size); // 构造函数

    ~chessboard(); // 析构函数

    void display(); // 打印棋盘格

    bool avail(int x, int y); // 判断落子合法性

    void go\_chess(int x, int y, char id); // 下棋

    bool judge(int x, int y, char id); // 判断是否连成五子

    void save(ofstream &outfile, const int &n, const int &num); // 保存棋盘

    void in(ifstream &infile, const int &n, int &num); // 读取棋盘

    void retract(); // 悔棋

    void evaluate(char id); // 评估玩家id合适的下棋位置

    int record\_num() // 返回当前储存的下棋顺序个数

    {

        return record.size();

    }

};

#endif

// CHESS\_H

chess.cpp文件

#include "chess.h"

#include <iomanip>

#include <windows.h>

// 构造大小为size的棋盘

chessboard::chessboard(int size)

{

    this->size = size;

    board = new chessnode \*[size];

    for (int i = 0; i < size; ++i)

    {

        board[i] = new chessnode[size];

        for (int j = 0; j < size; ++j)

        {

            board[i][j].set\_chessnode(i, j); // %player1  #player2

        }

    }

}

// 析构函数，释放内存

chessboard::~chessboard()

{

    for (int i = 0; i < size; ++i)

    {

        delete[] board[i];

        board[i] = nullptr;

    }

    delete[] board;

    board = nullptr;

}

// 打印棋盘格，为不同玩家设置了棋子颜色

void chessboard::display()

{

    cout << "   ";

    for (int i = 0; i < size; ++i)

        cout << setw(3) << left << i;

    cout << endl;

    for (int i = 0; i < size; ++i)

    {

        cout << setw(3) << left << i;

        for (int j = 0; j < size; ++j)

        {

            if (board[i][j].get\_signal() == '%')

            {

                SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_BLUE | FOREGROUND\_GREEN);

                cout << setw(3) << left << board[i][j].get\_signal();

                SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE);

            }

            else if (board[i][j].get\_signal() == '#')

            {

                SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_RED);

                cout << setw(3) << left << board[i][j].get\_signal();

                SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE);

            }

            else if (board[i][j].get\_signal() == 'o')

                cout << setw(3) << left << board[i][j].get\_signal();

        }

        cout << endl;

    }

}

// 判断(x,y)处落子合法性

bool chessboard::avail(int x, int y)

{

    if (x < 0 || x > 14 || y < 0 || y > 14)

    {

        cout << "请重新输入0~14?之间的正整数.";

        return false;

    }

    if (board[x][y].get\_signal() == 'o') // 出现异常

        return true;

    else

        cout << "该处已有棋子,请重新选择落子位置.";

    return false;

}

// 在(x,y)处下棋

void chessboard::go\_chess(int x, int y, char id)

{ // ????

    board[x][y].set\_signal(id);

    record.push\_back(board[x][y]);

}

// 悔棋，每次悔棋自动退回两步，即回到当前玩家所在的上一回合

void chessboard::retract()

{

    vector<chessnode>::iterator tmp = record.end() - 1;

    board[tmp->get\_x()][tmp->get\_y()].set\_signal('o');

    record.pop\_back();

    tmp = record.end() - 1;

    board[tmp->get\_x()][tmp->get\_y()].set\_signal('o');

    record.pop\_back();

}

// 保存棋盘

void chessboard::save(ofstream &outfile, const int &n, const int &num)

{

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            outfile << board[i][j].get\_signal() << endl;

        }

    }

    outfile << num << endl;

    outfile.close();

}

// 读取记录，并输入到当前棋盘中

void chessboard::in(ifstream &infile, const int &n, int &num)

{

    char ch;

    int x;

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            infile >> ch;

            board[i][j].set\_signal(ch);

        }

    }

    if (infile >> x)

        num = x;

    infile.close();

}

int chessboard::value(int x, int y, char id)

{

    /\*\*

     \* @brief  判断连子长度

     \* @param  x 棋子横坐标

     \* @param  y 棋子纵坐标

     \* @param  id 即假设该位置棋子为id

     \* @param  sum 分解每个方向上棋子连成的长度

     \* @return  max 该点所能连成的最大长度

     \*/

    int sum = 1, i = 1, j = 1, max = 0; // 自身点不用算，故都设置为1

    // 纵向

    while (x - i >= 0)

    {

        if (board[x - i][y].get\_signal() == id)

            ++sum;

        else

            break;

        ++i;

    }

    while (x + j <= 14)

    {

        if (board[x + j][y].get\_signal() == id)

            ++sum;

        else

            break;

        ++j;

    }

    if (sum > max)

        max = sum;

    // 横向

    sum = 1;

    i = 1;

    j = 1;

    while (y - i >= 0)

    {

        if (board[x][y - i].get\_signal() == id)

            ++sum;

        else

            break;

        ++i;

    }

    while (y + j <= 14)

    {

        if (board[x][y + j].get\_signal() == id)

            ++sum;

        else

            break;

        ++j;

    }

    if (sum > max)

        max = sum;

    // 主对角线

    sum = 1;

    i = 1;

    j = 1;

    while (x - i >= 0 && y - i >= 0)

    {

        if (board[x - i][y - i].get\_signal() == id)

            ++sum;

        else

            break;

        ++i;

    }

    while (x + j <= 14 && y + j <= 14)

    {

        if (board[x + j][y + j].get\_signal() == id)

            ++sum;

        else

            break;

        ++j;

    }

    if (sum > max)

        max = sum;

    // 副对角线

    sum = 1;

    i = 1;

    j = 1;

    while (x + i <= 14 && y - i >= 0)

    {

        if (board[x + i][y - i].get\_signal() == id)

            ++sum;

        else

            break;

        ++i;

    }

    while (x - j >= 0 && y + j <= 14)

    {

        if (board[x - j][y + j].get\_signal() == id)

            ++sum;

        else

            break;

        ++j;

    }

    if (sum > max)

        max = sum;

    return max;

}

// 判断是否连成五子

bool chessboard::judge(int x, int y, char id)

{

    int tmp = value(x, y, id);

    if (tmp >= 5)

        return true;

    return false;

}

void chessboard::evaluate(char id)

{

    /\*\*

     \* @brief  评估玩家id合适的下棋位置，遍历棋盘上所有为空的位置，计算假设落子后的连子长度，由大到小排序后输出(tip)

     \* @param  tmp 记录双方假设下棋后的连子长度

     \* @param  value1 储存value1>=3的点

     \* @param  value2 储存value2>=3的点

     \*/

    int tmp, i;

    for (int x = 0; x < 15; ++x)

    {

        for (int y = 0; y < 15; ++y)

        {

            if (board[x][y].get\_signal() == 'o')

            {

                // player1

                tmp = value(x, y, '%');

                if (tmp >= 3)

                {

                    board[x][y].set\_value1(tmp);

                    value1.push\_back(board[x][y]);

                }

                // player2

                tmp = value(x, y, '#');

                if (tmp >= 3)

                {

                    board[x][y].set\_value2(tmp);

                    value2.push\_back(board[x][y]);

                }

            }

        }

    }

    // 冒泡排序value1

    if (value1.size() != 0)

    {

        for (i = 0; i < value1.size() - 1; ++i)

        {

            for (auto p = value1.begin(); p != value1.end() - 1 - i; ++p)

            {

                if (p->get\_value1() < (p + 1)->get\_value1())

                {

                    chessnode chess = \*p;

                    \*p = \*(p + 1);

                    \*(p + 1) = chess;

                }

            }

        }

    }

    // 冒泡排序value2

    if (value2.size() != 0)

    {

        for (i = 0; i < value2.size() - 1; ++i)

        {

            for (auto p = value2.begin(); p != value2.end() - 1 - i; ++p)

            {

                if (p->get\_value2() < (p + 1)->get\_value2())

                {

                    chessnode chess = \*p;

                    \*p = \*(p + 1);

                    \*(p + 1) = chess;

                }

            }

        }

    }

    // player1

    if (id == '%')

    {

        if (value1.size() != 0)

        {

            cout << "如果您想进行攻击,可以选择以下位置:" << endl;

            for (auto p = value1.begin(); p != value1.end(); ++p)

                cout << '(' << p->get\_x() << ',' << p->get\_y() << ')' << endl;

        }

        else

            cout << "没有建议的进攻位置." << endl;

        if (value2.size() != 0)

        {

            cout << "如果您想进行防守,可以选择以下位置:" << endl;

            for (auto p = value2.begin(); p != value2.end(); ++p)

                cout << '(' << p->get\_x() << ',' << p->get\_y() << ')' << endl;

        }

        else

            cout << "没有建议的防守位置." << endl;

    }

    // player2

    else if (id == '#')

    {

        if (value2.size() != 0)

        {

            cout << "如果您想进行攻击,可以选择以下位置:" << endl;

            for (auto p = value2.begin(); p != value2.end(); ++p)

                cout << '(' << p->get\_x() << ',' << p->get\_y() << ')' << endl;

        }

        else

            cout << "没有建议的进攻位置." << endl;

        if (value1.size() != 0)

        {

            cout << "如果您想进行防守,可以选择以下位置:" << endl;

            for (auto p = value1.begin(); p != value1.end(); ++p)

                cout << '(' << p->get\_x() << ',' << p->get\_y() << ')' << endl;

        }

        else

            cout << "没有建议的防守位置." << endl;

    }

    // 清空value1,value2,留下次计算使用

    value1.clear();

    value2.clear();

}