****

高级程序设计实验报告

|  |  |
| --- | --- |
| 题 目： | 期末大作业 |
| 学生姓名： | 刘芳新 |
| 学 号： | 21312482 |
| 指导教师： | 冯国栋 |
| 专业班级： | 智科2班 |

2022年12月

本科生院制

目录

[1.综合概述 3](#_Toc23623)

[2.项目介绍与使用指南 3](#_Toc22490)

[3.实现方式综合概述 4](#_Toc15037)

[4.各类实现方式具体描述 5](#_Toc5680)

[4.1 Point类 5](#_Toc5680)

[4.2 Array类 5](#_Toc10422)

[4.3 chess类 6](#_Toc5680)

[4.4 kong类 6](#_Toc10422)

[4.5 卒类 6](#_Toc10422)

[4.6将类、车类、士类、炮类、象类、马类 7](#_Toc5680)

[4.7 棋盘类 7](#_Toc10422)

[4.7.1 Checkerboard[11][10][2]数组 8](#_Toc10422)

[4.7.2 Array<Array<chess\*>> allchess 8](#_Toc12020)

[4.7.3 initial\_walk与unitial\_walk函数 9](#_Toc7548)

[4.7.4 静态数据成员checker 9](#_Toc24076)

[4.7.5 构造函数 9](#_Toc2072)

[4.7.6 初始化函数initialization 1](#_Toc22059)0

[4.7.7 显示棋盘函数show 10](#_Toc6392)

[4.7.8 win函数与check函数 10](#_Toc21617)

[4.7.9 walk与move函数 10](#_Toc21617)

[4.7.10 Standard\_chess\_game标准棋局对战函数 11](#_Toc13724)

[4.8可读棋盘类 1](#_Toc5680)1

[4.8.1 构造函数（读取函数） 11](#_Toc14792)

[4.8.2 保存函数 11](#_Toc20200)

[4.8.3 标准对战与残局对战函数 12](#_Toc8330)

[4.9 main函数 12](#_Toc22222)

[5.总结与心得体会 12](#_Toc22515)

**1.综合概述**

本次大作业内容完成于11月29日至12月4日，均本人手打原创代码，除Array类借鉴并改编了往次实验课自行实现的内容外，所有类与函数均于这段时间内自行构思并实现。

本项目包含了12个头文件和10个源文件，代码行数（不包括空白行和注释行）如下表所示：



本项目实现了“C++期末大作业”中的所有要求，并具有明显有用功能、意义（实现双人对战中国象棋游戏）。

本项目压缩包“中国象棋.zip”，内含“code”文件夹中，存放了所有源代码文件。

**加分项：**本人做了简单的应用打包，做了一个安装与卸载程序，即压缩包中的“chess.msi”与“setup.exe”，按照“安装指南.txt”中的指示按照即可在电脑上安装使用本项目。

**2.项目介绍与使用指南**

本项目如名，是一个实现中国象棋游戏的项目。关于中国象棋游戏，其规则如“rule.txt”文件中所述，包括了中国传统象棋游戏的所有规则，此处不再赘述。

本项目所实现的游戏中，红黑方用不同的中国汉字表示，同时对不同类型棋子进行编号，如“index.txt”文件所示。游戏中将象棋的棋盘编写了行号与列号，游戏以吃掉对方的“将”（“帅”）或逼迫对方投降为目的，操作方式为：按照窗口输出的提示，当前轮到的红或黑方一次性输入4个数字，分别代表棋子编号、第几个该类型的棋子、移动后该棋子的行号、移动后该棋子的列号（如：红方想把“俥1”移到4行5列位置，输入3 1 4 5即可）。执行命令之后窗口会将更新后的棋盘输出。此时另一方再执行自己的命令......

程序内定义了标准棋局，即按照中国象棋游戏的标准开局，创建一盘新的游戏。同时程序提供了读取与保存功能，用来读取残局与保存残局（保存精确到红、黑方的出手回合）。残局文件的格式如压缩包中“标准残局文件.txt”所示，此处不再贴图。

若在红黑方输入命令环节，未按照标准输入，会有以下情况：

若输入index，则输出棋子编号对照表，之后回到输入命令环节，仍是当前红、黑方的回合。注意，对于用户而言，棋子编号仅与棋子种类有关，与红黑方无关，如红黑方的“车”都是3号。

若输入0，则本次出手的红、黑方会投降，另一方获胜，并结束当前游戏。投降操作具有重大意义，由于象棋中“将军”机制的存在，“将”（“帅”）一般不会被吃掉，传统象棋中，判断某方获胜的标准为“绝杀”或“困弊”。“绝杀”指的是对方使用“将军”操作，将己方的“将”逼迫得无路可走（不管往何处走都无法脱离被“将军”）。“困弊”指的是对方未“将军”，但是轮到己方回合时，己方已无棋可走（不管移动哪一个棋子，都会主动使自己被“将军”）。此两种情况，本程序不会直接进行判断，但是遭遇这两种情况时，用户将无法移动自己的任何棋子，逼迫用户只能以输入0的方式主动认输。

若输入save，则会保存当前残局，但当前游戏仍会继续。多次保存不会覆盖原文件。

更多介绍请参见“read me！！！.txt”文件。

**3.实现方式综合概述**

象棋包含了7种棋子，共32个棋子，其中红黑方各5个卒、1个将、2个车、2个士、2个炮、2个象、2个马。象棋的棋盘可以抽象成为一个10行9列的数组，将不同种类棋子编号，将无棋子的空格编号为0，即可用一个10\*9的二维数组表示整个棋盘。考虑到红黑方的不同，所以7种棋子应当编成14种号码（程序内实现用14种，为了方便用户，用户只需考虑7种即可）。

同种类棋子有重复的情况，如卒有5个。为了加以区分，将其分别再标号1-5，在命名时加以区别，同时在棋盘上也通过名字不同加以区别。

为实现上述思路，首先应当定义一个“chess”类，用来概况所有棋子的共性，然后由chess类分别继承派生出“卒”、“将”等具体的棋类，里面包含了它们独特的命名、编号与移动规则。

同时，由于“卒”、“将”等棋类都是独立的，而象棋规则中有一些与其他棋子联动的移动规则，在这些棋类中无法定义完全象棋规则。我们再定义一个“棋盘”类，用来储存32个棋子与它们的属性，并将之前所提及的二维数组也一并包括在内，这样便可借助棋盘来定义那些联动规则。

“棋盘”类已实现中国象棋游戏的所有基本操作，在此基础上，为了实现读取与保存功能，从“棋盘”类中继承派生出“可读棋盘”类。实际上是间接控制着“棋盘”类，运行方式还是跟棋盘类一样，只不过构造函数中用读取的文件的内容来初始化棋盘。

此外，Array类是用来替代vecter功能的类，与vecter的功能并无大的区别。

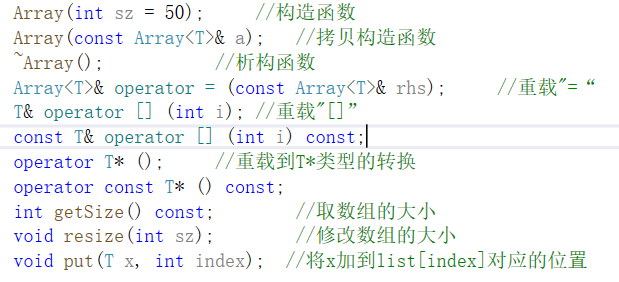
**4.各类实现方式具体描述**

**4.1 Point类：**

点类是用来描述棋子在棋盘中的位置的，由于位置是整数，所以包含两个int型的私有数据成员x和y。提供了=运算符重载和==、！=的布尔型运算符重载。

**4.2 Array类：**

动态数组类是一个template模板类，使用方式与内容均与vecter类一样，此次不再赘述。其内函数实现了“=”、“[ ]”等运算符的重载，还有数组大小“size”的重构等。具体函数种类如图所示：



**4.3 chess类：**

chess 类是所有棋类的基类，不允许随意使用，故定义为抽象类，并将其析构函数定义为虚函数。其内包含了5个私有数据成员，分别为string chessname（用来存放棋子名），bool color（用来标记该棋子属于红、黑方），int index（棋子编号），bool alive（用来标记该棋子是否存活，被吃掉的其中不存活），Point position（用来存放棋子的坐标）。

chess类内部除了基本的构造、析构、复制构造和运算符重载，以及获取、修改私有数据成员的函数以外，最重要的有两个纯虚函数：bool move(Point Position)，用来判断是否能移动到Position的坐标；void set(bool Color)，用来初始化颜色，对于具体的棋类来说，只要颜色确定，其除了坐标以外的任何属性都能确定。

**4.4 kong类：**

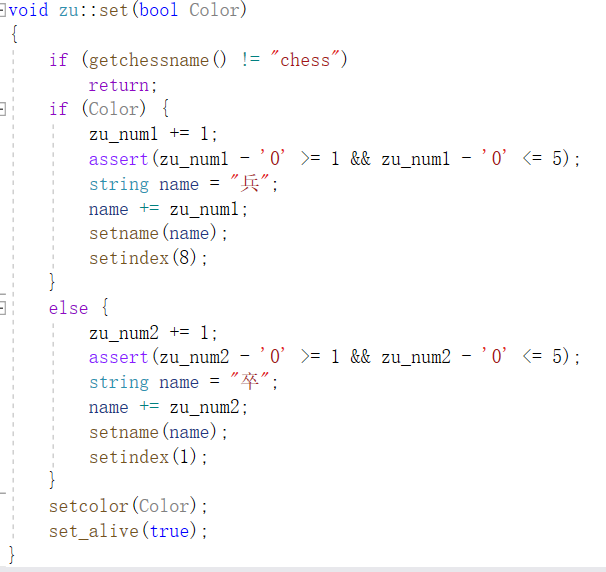
kong类与chess类一起定义在“chess.h”中。由于chess类是抽象类，不可实例化对象，而棋盘中存在空格，于是需要一个kong类。它虚继承自chess类，用来表示棋盘中的空格。它的move函数直接返回false，因为空格不可移动。

**4.5 卒类：**

卒类虚继承自chess类，其内部包含两个char型静态数据成员nums，分别储存红黑方的卒的数量。这个数量分别不可少于0个，不可多于5个，否则会通过assert报错。这两个数据成员之所以定义为char类型，是因为它们还参与棋子的命名，每通过color定义一个卒，其对应的nums便会+1，然后直接加入其名字的字符串后面。如第三个定义的红色卒，会在原本的命名“兵”后直接加上“3”，最终名字为“兵3”。这也是同颜色卒的数量只能控制在0到5的原因：不允许命名为“卒6”之类的违反规定的名字。

其move函数是卒的核心函数，规定了卒的移动方式（只能移动一格，不可后退，过河前不可左右移动），若不符合规定的移动，move将返回false。具体实现方式是根据“color”和“position”来分别考虑的，此处不赘述。

其set函数是对构造函数的补充，在set函数执行后该卒自动被命名，同时一些属性也因此改变，因此set函数不可轻易动用。具体实现方式如下：

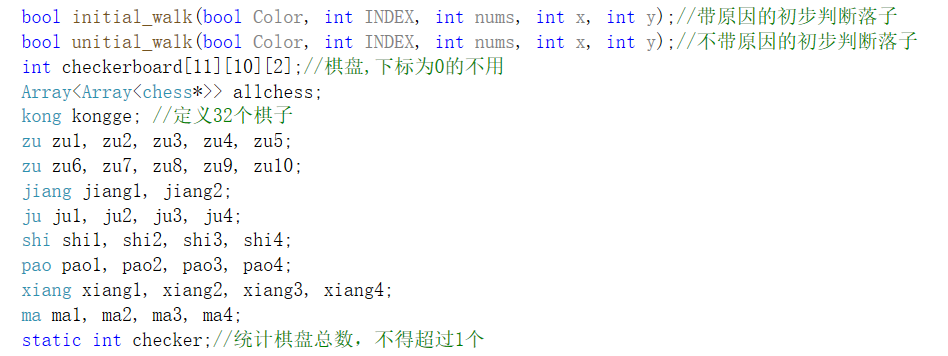


**4.6 将类、车类、士类、炮类、象类、马类：**

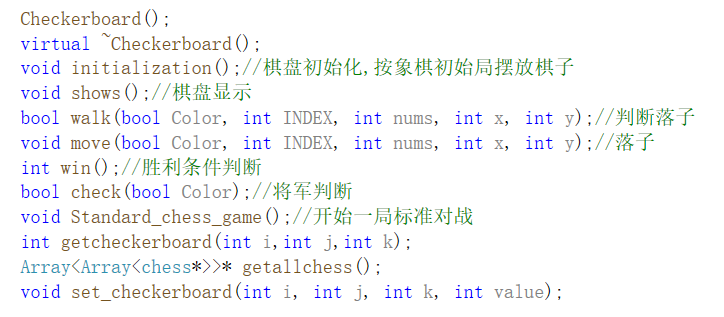
这些类与卒类非常类似，其内部实现的内容也与卒类大同小异。每个类内部都定义了不同的nums，但实现的功能与卒类中的nums是一样的。最大的不同是各自的move函数都非常不同，这是根据它们彼此完全不同的移动规则来决定的，这里也不赘述。

**4.7 棋盘类：**

棋盘类是整个程序的核心，也是最复杂的部分。其私有数据成员如下：



其公有函数成员如下：



4.7.1 Checkerboard[11][10][2]数组

Int型三维数组Checkerboard[11][10][2]是棋盘，前两个维度中下标为0的不用，而第三个维度的第一个格子用来储存该位置的棋子编号（index），第二个格子用来储存这是第几个该棋子（nums）。如“卒5”的index是1，其nums是5。

4.7.2 Array<Array<chess\*>> allchess

allchess是用来存放所有棋子信息的动态数组，它的每一个元素都是一个储存着若干chess型指针的动态数组。allchess[i]储存的就是index为i的所有棋子的信息，而allchess[i][j]就是一个指向index为i，nums为j的棋子的指针。通过Checkerboard数组与allchess的配合双向对应，就可以实现“将棋子摆上棋盘”的行为。对于Checkerboard数组上储存的信息，可以通过查询allchess来获取这个棋子的信息，从而判断下一步的操作。

4.7.3 initial\_walk与unitial\_walk函数

initial\_walk与unitial\_walk函数是实现2.中所述的命令输入环节的至关重要的函数，用来检测某个棋子移动到某个位置的操作是否合理。其差别为：initial\_walk函数会输出不合理的原因，而unitial\_walk函数则不会。

在具体的实现中，这两个函数会根据参数列表的前三个参数来确定要移动的是哪个棋子，然后根据Checkerboard数组以及allchess进行移动判断。

这个判断分两步，第一步是移动前，首先否决掉不符合对应棋子move函数的操作。然后再考虑这个移动是否正常，如不能原地不动，不能吃掉自己的棋子，死亡（alive==false）的棋子不可移动等。最后根据棋子类型，进行一些联动规则判断，如马会卡马脚，炮需要隔一个炮台，象会卡象脚等。

第二步判断是在第一步判断成立的基础上，用回溯法判断移动后是否合理：将棋子移动过去（若移动后位置有敌方棋子则将敌方棋子移走暂存），然后判断此时会不会给己方造成危险，如是否满足“飞将”的条件，是否会导致自己被“将军”等。判断完毕之后，不管结果如何，都要将“移动”给撤销，因为这个函数只是判断函数，不是操作函数。

4.7.4 静态数据成员checker

设置这个int型数据是由于各棋子都不能超过其规定的数量，且“可读棋盘”类也是在同一个棋盘上进行操作，所以定义两个棋盘的操作会失败。

4.7.5 构造函数

由于类定义数据成员不能初始化，构造函数需要初始化各棋子以及allchess。首先调整allchess的size为15（0号存空格，1至14号存棋子），然后根据各类型棋子数量调整allchess[i]的size。然后对于定义好的32个棋子，以及定义的空格，将它们的地址存入allchess中，这样完成了allchess的初始化。

最后，遍历allchess，通过棋类中的set函数，给每个棋子设置颜色，这样其名字与编号也会自动建立，所有棋子就初始化完毕了。

4.7.6 初始化函数initialization

这是将棋盘摆放到标准棋局的函数。首先将Checkerboard元素全部设置为0，清空棋盘。然后将allchess中的棋子的“alive”属性全部设置为true。最后根据标准棋局，手动调整Checkerboard中的数以及设置allchess中棋子的position，实现“摆放好棋子”的功能。

4.7.7 显示棋盘函数show

show函数通过遍历Checkerboard数组的方式，将数组中每个格子的棋子都打印出来，空格不打印，同时编写行号与列号，这样子可以实现对棋盘上局面的一个较为美观的显示。

4.7.8 win函数与check函数

Win函数是胜利判断函数，用来判断场上将、帅的存活。由于象棋胜利一般为绝杀、困弊机制，所以win函数生效的场景较少。Check函数是“将军”函数，带一个参数color，用来判断color对应的这一方是否处于被“将军”的状态。判断方法为：遍历敌方所有存活的棋子，用unitial\_walk函数判断这些棋子是否能移动到己方“将”的位置。

4.7.9 walk与move函数

Walk函数是initial\_walk函数的改进，比起initial\_walk函数，它新增了判断是否“将军”的功能，会对被“将军”的一方发出警告。Move函数则是单纯的移动棋子，直接改变棋子和棋盘属性，将棋子移到规定的位置。Move函数需要配合上walk函数使用。

4.7.10 Standard\_chess\_game标准棋局对战函数

这个函数整合上述所有函数，并规范了流程，先用initialization函数将棋盘初始化，再用while循环使红、黑方轮流行动，行动时有命令输入环节，得到指令之后，会先判断这个指令是投降指令，或者查看棋子编号指令，然后再将其作为移动棋子指令，用walk函数与move函数实现。移动后会用show函数再次打印棋盘，并通过check函数进行“将军”判断，若被“将军”则会发出提示。

**4.8 可读棋盘类：**

可读棋盘类虚继承自棋盘类，是对棋盘类的补充。可读棋盘类补充了根据文件路径初始化棋盘的构造函数，以及残局对战、保存残局的函数。同时，它有两个私有数据成员，私有静态数据成员times是在保存函数中为保存文件命名的函数，以实现保存文件不相互覆盖；color则是保存文件的先手方颜色，这是因为残局不像标准棋局那样固定由红方先手，所以需要一个先手方的颜色。

4.8.1 构造函数（读取函数）

构造函数实现了读取函数的功能，它在棋盘类构造函数建立好空的棋盘后，读取文件，从文件中提取棋子信息，并在把棋子摆放到棋盘上。同时，它会对读取的内容进行判断，如果读取的文件格式不对、某棋子的数量超过象棋规定、某棋子的位置在象棋中不可能实现，它会停止读取，输出错误原因，并用initialization函数将棋盘初始化，中断残局游戏，转而进行标准棋局游戏。

4.8.2 保存函数

保存函数会先将times加一后转化成字符串类型，然后用相对路径，在本程序安装目录中的images文件夹内创建新的txt文件，并用times将其命名。接着遍历棋盘，输出棋子的位置。

4.8.3 标准对战与残局对战函数

标准对战函数在棋盘类的标准对战函数基础上重构，新增了输入save保存当前棋盘的功能。残局对战函数比起标准对战函数，少了初始化棋盘的功能，这样便可在读取后摆放好的棋盘上对战。

**4.9 main函数**

本程序中大部分操作都定义在各类中，main函数仅是补充一些细枝末节，并让用户选择残局对战还是标准棋局对战。

**5.总结与心得体会**

本程序虽然实现时间较短（只有5天），却是本人构思数星期的成果。在实现各类的过程中不断遇到问题，然后解决问题，不知不觉代码就超过2000行。除了此象棋程序外，本人的多个构思均无法同时使用到“C++期末大作业”中规定的所有知识点，若是刻意使用则会影响程序美观与简便性，本末倒置。

比起Python语言，C++没有那么便利，但是运行效率更快，而且也更有一种“精妙”的感觉。虽然许多Python很容易实现的操作，在C++中需要通过巧妙的方法与技巧才能实现，但是在这个过程中有一种摆弄精巧玩具的感觉，也是C++独特的魅力。