chess4 帮助及说明

在为我的作业评分之前，请您花两分钟时间阅读这份文档。

[特别注意] 这份 C 语言作业涉及了部分工程性的代码编写方式。如果您看不懂我的部分或所有代码，请您向下滚动，直接阅读 `如果我看不懂这份作业该怎么办？` 一节。

无论如何，请您运行 `chess4.exe` 体验一下程序质量再进行评分。

内容：

代码阅读的注意事项

这份作业包含三个版本。**三个版本的源代码是相同的。**

- 如果您想使用 Code::Blocks 查看这份作业的代码，请把 `CodeBlocks 版工程` 文件夹下的 `chess4.cbp` 文件用鼠标拖拽到您的 Code::Blocks 窗口里。
- 如果您想使用 Dev-C++ / Vim / Emacs / Sublime Text 等工具查看代码，我建议您浏览 `单文件版` 文件夹下的代码。

    - 如果您使用 **Code::Blocks**，也可以由您手动创建一个新项目，并使用此版本代码复制到您项目的 `main.c` 内。构建说明见文档结尾「构建」一节。

- 如果您安装有 **Visual Studio Code 和 git**，我且假设您能看懂我在说什么。请您：

    - 您可以使用 Visual Studio Code 配合 git 阅读源代码。

    - **强烈建议** 阅读下面的 `构建说明` 章节并尝试构建这份代码。

所有函数的声明处（`.h` 文件）都写有注释。您也可以对照作业报告内的表格查看函数的用途。

如果我看不懂这份作业该怎么办？

为了方便您对这份作业进行评分，也为了贯彻落实 *面向评分标准编程* 的理念，下面我**直接列出代码中对应评分标准要求之处。**

> 如果您想查验，请您打开 **「单文件版」** 文件夹下的代码对照。

数据结构部分

要求 1：数组 / 结构体数组 / 结构体数组+指针

见代码第 `11` 行：`struct _HistoryA` 的声明。这个结构体内含有**数个不同类型的指针。**

见代码第 `22` 行：`struct _Board` 的声明。这个结构体内含有

```
1. 数个不同类型的指针，包括一个整型二维数组的指针。
2. 一个指向其他结构体的指针。
```

要求 2：链表等动态数据结构

见代码第 `51` 行：`struct _FileLL` 的声明。这是一个**链表结构体。**

算法部分

要求 1：枚举/递推/迭代/分类统计

见代码第 `614` 到 `684` 行：`int Finish(Board *bd)` 函数。

> 此函数运用了枚举和分类统计法，判断了游戏局面的结束与否。

要求 2：排序/查找

见代码第 `155` 到 `174` 行：`int Get_single_key_input(char *req)` 函数。

> 此函数运用了查找法判断读入字符与字符串的匹配。

见代码第 `293` 到 `323` 行：`FileLL* Resolute_logtree(FileLL *head)` 函数。

> 此函数运用了 `sprintf` 函数和记录解析方法查找历史记录文件。

要求 3：文件操作

见代码第 `222` `234` `300` `314` `496` 行及其前后。

> 程序中的**历史记录系统**会将历史记录保存在文件中。设计了文件读写。您也可以运行 `chess4.exe` 并查看它生成的 `Histories` 文件夹及其一系列子文件来验证这一点。

要求 4：模糊匹配

见代码第 `155` 到 `174` 行：`int Get_single_key_input(char *req)` 函数。

> 此函数支持插入一个 **「通配符」** `_` 符号来匹配模式串中未出现的字符。实现了模糊匹配功能。
>
> 您也可以查看函数声明处的注释，结合函数代码验证（`70` 行处）。

要求 5：递归

见代码第 `325` 到 `331` 行：`void Destroy_filelog(FileLL *head)` 函数。

> 此函数运用递归销毁链表并回收空间。

程序设计质量部分

模块化设计

请您打开 `原始工程` 文件夹，并浏览程序是如何被划分成多个文件，分别管理的。

您可以查看代码开头各个函数的声明，来了解函数参数的简洁性和独立性。

防御式编程

函数 `Get_single_key_input()` 及 `ReadInt()` 可以处理所有非法输入。您无法对程序内部的函数输入非法数据。

程序有容错能力。比如您无法在满格列上落子，您也无法把光标移出棋盘。

代码规范

您可以打开 `单文件版` 代码查看完整的代码缩进和统一的命名风格。

如，函数使用 snake case with capital 命名法则，即：首字母大写，词之间添加下划线。

用户界面友好

请您运行程序并体验。

使用了新颖独特的设计

输入采用非阻塞式即时响应方法。我认为这是新颖的设计。

请您运行程序并体验。

构建说明

单文件版

您可以直接使用您的 **Dev-C++** 等 IDE 编译运行。

如果您配置了编译器，以 MinGW 为例，在命令行输入：

`gcc main.c -O2` 生成可执行文件。

Code::Blocks 工程版

打开工程之后，请您使用 Code::Blocks 构建。

按照原始工程方式构建

我默认您可以读懂以下文字，并在 Windows 操作系统下：

请您确保安装了 **3.2** 及以上版本的 CMake，7.0 以上版本的 MinGW-W64 编译器，并全部配置正确。

请您打开 MinGW-W64 的 `\bin` 文件夹，将 `mingw32-make.exe` 文件复制一份并重命名为 `make.exe`。

请您先单独构建 `Natsu` 库。切换到 `\src\natsu` 目录，在命令行输入：

```
cmake . -G"MinGW Makefiles"
```

```
make
```

然后回到 `\src` 目录，构建整个程序。在命令行输入：

```
cmake . -G"MinGW Makefiles"
```

```
make
```

您会看到数个绿色的输出，左端有类似 `[66%]` 的进度提示。当提示达到 100% 时，构建完成。构建的可执行文件位于 `\bin` 目录下。

4. 趣事

本项目在 Github 上创立了 Repository，评分结束后公开，公开后您可以通过搜索代码内容找到。您可以去观光或添加一个 star。（感谢！）

您可以阅读 `原始工程` 下的 `Readme.md` 文档了解我是如何完成这份作业的。

感谢您阅读这份文档。

GNAQ | 2020/12/31 02:15