Pacman!

Pacman （吃豆人）是一个简单有趣的小游戏。如果你没玩过，可以用 Google 搜索“pacman”，跳出来的那个在线小游戏就是。

这次作业中，你将要用 C 语言实现一个**命令行** (command-line) 的 Pacman 游戏。作为参考，你可以在 `demo` 文件夹下找到我们实现好的版本。

Pacman 的规则非常简单：局面上有若干食物 (Food) 、鬼 (Ghost) 和一个 Pacman 。玩家需要移动 Pacman （用 w/a/s/d 键控制），在避开鬼的情况下吃掉所有食物。 Pacman 碰到鬼则失败，吃完所有食物则胜利。具体的分数计算方式、鬼的移动方式等等有很多变种，在我们的实现中

- 每 0.5s 所有的鬼都会移动一步，分数会减少 1 。
- 每吃到一个食物，分数增加 10 。
- 鬼会朝着离 Pacman 更近的方向移动。

注意，你不需要和我们使用相同的策略，但是你得使用一个明确的、合理的策略。其它各个方面也是如此。

运行参考程序

我们提供 Linux, Mac OS, Windows 三种平台上的参考程序可执行文件，它们都位于 `demo` 文件夹下。为了方便描述，假设这个可执行文件的名称是 `pacman` 。

运行的方式有两种：

1. 无参数，直接 `./pacman` 或者双击图标运行，效果是从第 0 关开始，每通过一关可以进入下一关。
2. 带一个整数参数 `n` ，表示运行第 `n` 关。例如 `./pacman 3` 就是玩第 `3` 关。

在终端运行时，需要保证 working directory 是 `demo` 。对于 Mac OS ，你需要先 `chmod +x pacman_macos` 为 `pacman_macos` 赋予执行权限。

关卡地图

我们准备了 4 个关卡的地图，它们以文本文档的形式存放于 `maps` 文件夹下。一个关卡地图文件的格式是

- 第一行，三个整数 r, c, g 表示该关卡的地图有 r 行、 c 列，有 g 个鬼，其中 $g < 10$ 。
- 接下来是一张 r 行 c 列的地图，用 $r \times c$ 个字符表示其中每个位置的物体。
    - 空格 ` ` 表示该位置没有物体。
    - 点 `.` 表示该位置有一个食物。
    - `C` 表示该位置是 Pacman 。保证整张地图有且只有一个 `C` 。
    - 数字字符 i 表示该位置是 i 号鬼，其中 $0 \leqslant i < g$ 。
    - `R`, `Y`, `G`, `B` 分别表示该位置是红色、黄色、绿色、蓝色的墙。
- 接下来 g 行，其中第 i 行是一个字符串 `up` 、 `down` 、 `left` 或 `right` ，表示 i 号鬼的初始运动方向（见下文有关鬼的移动的描述）。

这些地图是可以改的。你可以想办法支持更多的鬼，或者允许不同的鬼具有不同的速度等。

代码文件

代码分为两个文件，分别是 `utils.h` 和 `pacman.c` ，其中前者被后者 `#include` 。

`utils.h` 包含各种和系统有关的函数，例如隐藏光标、移动光标、清屏、控制输出颜色、获取键盘动作。你不需要改动这里的内容，但如果实在想改也是可以的，比如你可能想用一些别的颜色？

`pacman.c` 是主文件，包含了整个游戏的实现。稍后我们会仔细介绍其中的内容和你要完成的部分。

交互

我们采用**实时交互**的方式。每隔一定的时间（在我们的实现中为 20ms），程序就会检测一下是否有键盘的动作，并作出相应的反应（相关函数为 `runGame` 和 `movePacman` ）。检测键盘是否有键被按下的函数是 `kbhit()` ，读取被按下的键的函数是 `getch()` 。

在 `printInitialGame` 之后，屏幕上的内容将会不断地被**更新**。 `move_cursor(row, col)` 函数可以将光标移动到第 `row` 行、第 `col` 列，这里的行号和列号从 `0` 开始。

注意，这种交互方式对效率是有要求的。移动一个物品应该只涉及到一两个字符的更新，更新一个字符只需将光标移动到相应的位置并输出新的字符。**每一帧都清屏并重新输出全部内容的做法是不可接受的**。当然，其它的函数也需要高效的实现，例如 `moveGhosts` 和 `movePacman` 都不能遍历整个 `grid` 。

总体要求

本题采用和 TA 面对面 check 的方式来评分， OJ 上不设测试，**但是你需要提交你的代码**（见提交方式）。你和 TA 当面 check 的代码应当是你在 OJ 上的最后一次提交。

在游戏的实现上，我们允许你有较多的自由。你可以自行决定很多东西，例如鬼的移动速度、计分方式、显示的字符的颜色、关卡运行模式等，但你需要有一个明确的、合理的决定，并正确地实现出来。

我们提供的代码框架、地图文件等都是可以改的。如果有好的理由，你完全可以重新设计地图文件的格式、交互的逻辑等等。

代码本身的质量也是评分项，而非仅仅是“能跑就行”，因为追求高质量的代码也是编程学习中十分重要的一环。你的代码需要是格式化的，并且经过了良好的组织，不随意使用全局变量，变量在即将被使用的时候才被声明，遵守一致的命名规范，避免大段的雷同，避免内存泄漏，在 GCC 的 `-Wall -Wpedantic -Wextra` 编译选项之下没有 warning 等。特别地，对于 `struct` 对象的初始化和赋值应尽可能多用 initializer-list （包括 designators）、 compound literals 等语法，它们比罗列一条一条的赋值语句更简洁、更清晰。

阅读代码的能力也在本题的考查范围之内。你需要读懂 `pacman.c` 中的每一个部分，包括我们已经实现好的功能，这当然也是有助于你正确地实现你的部分的。我们会在 check 时问及其中的一些细节。

另外，我们在代码中以注释的形式给出了很多的提示，在你实现了相关函数之后这些注释应该被删除或作相应的调整。特别地， `FIXME:` 和 `TODO:` 是代码项目中常见的标注，现代编辑器一般都可以将其高亮显示（例如 VSCode 的 `TODO Highlight` 插件），它们通常表示某个功能的实现有问题、需要更正或者完善。在相应的问题被解决之后，你应该将 `FIXME:` 或 `TODO:` 改为 `DONE:` ，或直接将那条注释删去。

提交方式

将 `maps` , `pacman.c` , `utils.h` 打包为一个 zip 文件提交到 OJ 。如果你还增加了其它文件，也需一并打包提交。

具体要求

注意，以下是你需要完成的五个部分，但并不意味着你只需要阅读这五个部分的代码。你要做的是“让游戏跑起来”，而不是“完成一个函数”。诸如“某个函数需不需要做某件事”这样的问题，你应该根据题目描述和代码前后的逻辑自行得出答案，我们不会再回答。

1. 实现 `createGame`

函数 `Game createGame(int level, const char *mapFileName)` 创建一个 `Game` 对象，关卡编号是 `level`，关卡的信息存放于 `mapFileName` 文件中（见关卡地图）。你需要正确地设置这个 `Game` 包含的所有信息，特别是

- `status` 应被设为 `GS_Preparing`。
- `score` 应被设为零。
- 每一个鬼的 `itemBelow` 应当被设置为一个合理的值，这个值会和你如何移动鬼有关。
- `grid` 应该是一个动态分配的“二维数组”。你可以阅读 `printInitialGame` 函数的定义来推测 `grid` 中应该存储什么样的内容。

不要忘记关闭你打开的文件。

注意，与之配套的 `void destroyGame(Game *pGame)` 函数负责释放一个 `Game` 对象所持有的资源，如果你为 `Game` 添加了一些东西，你可能还需要修改 `destroyGame`。

值得一提的是，`createGame` 是一个类似于 C++ 中的“构造函数”的函数，因为它负责创建一个对象并初始化它的成员。当前的设计中，`createGame` 创建出一个 `Game` 对象，将其初始化之后 `return` 出来。除此之外，可能的设计还有以下几种：

1. ```
   void createGame(Game *pGame, int level, const char *mapFileName);
   ```

   对 `pGame` 所指向的对象进行初始化。用户需要自行创建一个 `Game` 对象，并将其地址传进来：

   ```
   Game game;
   createGame(&game, 0, "maps/level0.txt");
   ```

2. ```
   Game *createGame(int level, const char *mapFileName);
   ```

   动态分配内存创建一个 `Game` 对象并初始化，返回这个对象的地址。这种设计下，`createGame` 创建的 `Game` 对象必然使用动态内存，与之配套的 `destroyGame` 则也需要 `free(pGame);`。

比较这三种设计的异同，分析它们的利弊。

2. 实现 `moveGhosts`

函数 `void moveGhosts(Game *pGame)` 将所有的鬼都移动一步。这些鬼的信息需要被更新，同时你还需更新 `pGame->grid` 中的相应内容以及屏幕上打印出来的内容。

鬼可能会有重叠，鬼也可能位于一个食物之上，你需要正确地处理这些情况。提示：

- 想一想怎么利用 `Ghost` 的 `itemBelow` 成员。
- 为了解决对象重叠的问题，一种可能的办法是先倒序清除所有的鬼，再正序将鬼放置在新的位置上。如果你用这种方法，你需要想清楚它为什么是对的。

在我们的实现中，鬼总是会朝着离 Pacman 更近的方向移动，但这可能需要一些算法知识。你可以只实现最基本的**来回移动**的鬼：它的初始方向由地图文件给出，按照此方向不断移动，当碰到墙壁或边界时则调转方向。当然，你也可以实现更加智能的移动策略，但是这不会带来加分。

移动鬼可能会杀死 Pacman 导致游戏结束，但请务必想清楚 `moveGhosts` 这个函数的职责究竟是什么，超出其职责范围的工作应该由其它的函数完成。

3. 实现 `movePacman`

函数 `void movePacman(Game *pGame)` 将 Pacman 移动一步，方向取决于用户的键盘动作（可能没有）。Pacman 的信息、 `pGame->grid` 中的内容以及屏幕上打印出来的内容需要得到更新。

你将会需要 `getPacmanMovement()` 函数，这个函数会读取用户的键盘动作，如果用户没有按下键盘或者按下的键不是 W/A/S/D 之一则返回 `Idle` 。

移动 Pacman 可能会杀死 Pacman 导致游戏结束，但请务必想清楚 `movePacman` 这个函数的职责究竟是什么，超出其职责范围的工作应该由其它的函数完成。

4. 实现 `pacmanDies`

函数 `bool pacmanDies(const Game *pGame)` 判断 Pacman 是否已经死了。这个判断应该非常简短、快速。

5. 关卡相关的功能

我们的参考程序会从第 0 关开始，每通过一关才进入下一关，若某一关失败还可以选择重新开始。你能实现这样的功能吗？怎样设计关卡之间与用户的交互？

你还可以尝试实现更复杂的功能，例如记录每一关的历史最高分、存档等等。但是更复杂的功能并不会带来加分。