数据结构实验一说明

1 概览

本次实验的内容是基于给定的框架，实现2048游戏的基本功能。游戏的核心理念是通过合并相同的数字方块，最终达到 2048 这一目标。其基本规则如下：

1. 棋盘： 4x4 的网格。
2. 初始状态： 随机生成两个数字方块（2 或 4）。
3. 移动： 玩家可以通过上下左右四个方向移动所有方块。
4. 合并： 当两个相同数字的方块碰撞时，它们会合并成一个新的方块，数值为原来两个方块数值之和。
5. 目标： 通过不断移动和合并方块，最终达到 2048。
6. 结束条件： 当棋盘满且无法进行任何合并时，游戏结束。
7. 分数计算： 每次合并方块时，玩家的分数会增加合并后方块的数值。

---

2 预备知识

2.1 C++ 基础

在开始实验之前，确保你对以下 C++ 基础知识有一定的了解。C++ 支持 面向对象编程（OOP），通过类（class）来封装数据和操作。类可以包含私有（private）、保护（protected）和公有（public）成员，提供了良好的信息隐藏机制。

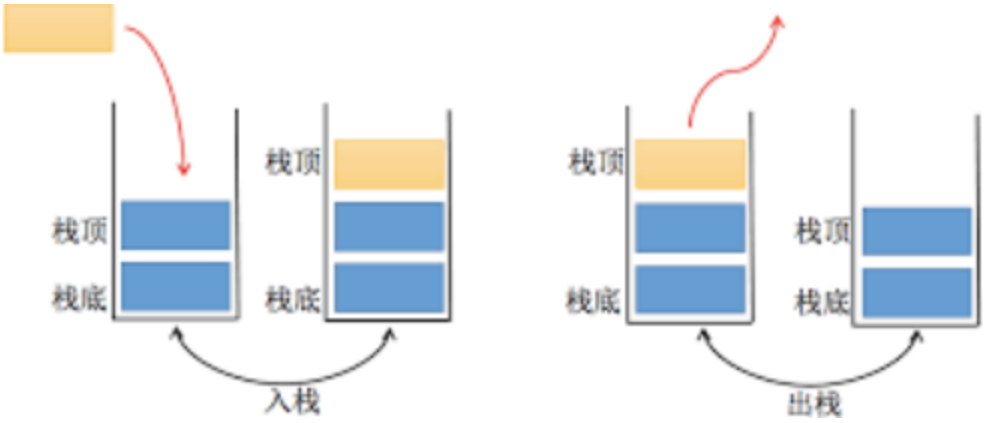
具体地，可参考C++ 教程进行学习。

2.2 线性表

本次实验将会采取二维数组来实现棋盘的存储。只不过在C++中，我们使用`vector<vector<int>>`来实现二维数组。二维数组的每个元素可以通过两个索引来访问，分别表示行和列。其中，初始化等操作以完成，因此你只需要了解二维数组的基本操作即可。

2.3 栈

本次实验存在悔棋操作，对于悔棋操作，我们可以使用栈来存储每一步的棋盘状态。栈是一种后进先出（LIFO）的数据结构，支持两种基本操作：入栈（push）和出栈（pop）。在C++中，我们可以使用`std::stack`来实现栈。



具体的栈操作可以参考栈的介绍。

---

3 实验框架

实验框架已经为你提供了基本的类和函数定义，你需要在指定的位置实现相应的功能。

代码框架如下：

```
Lab1/
├── include/
│   ├── Game.h                    # 游戏状态控制
│   ├── GameBoard.h               # 棋盘管理
│   ├── GameRender.h              # 渲染和显示
│   └── InputHandler.h            # 输入处理
├── src/
│   ├── Game.cpp
│   ├── GameBoard.cpp
│   ├── GameRender.cpp
│   ├── InputHandler.cpp
│   └── main.cpp                  # 程序入口
├── CMakeLists.txt                # 构建项目，生成可执行文件
├── Guidelines-lab1.pdf           # 实验说明
└── Windows项目构建步骤说明.pdf
```

4 实验任务

首先，我们约定：

1. 棋盘的左下角为 (0, 0)，棋盘在第一象限，横轴为 x，纵轴为 y。
2. tile 这个词为一个术语，表示 2048 棋盘中任何一个 cell 的值。

4.1 GameBoard.cpp

1. `slide` 滑动函数

本函数实现棋盘上所有方块的滑动逻辑，滑动方向包括上、下、左、右四个方向。滑动时，所有方块会尽可能地向指定方向移动，直到遇到边界或其他方块为止。已完成向上滑动的逻辑，你需要实现其他三个方向的滑动逻辑。

```
void GameBoard::slide(Direction direction) {
    switch (direction) {
        case Direction::UP:
            for (int col = 0; col < 4; ++col) {
                int target = 3;
                for (int row = 3; row >= 0; --row) {
                    if (board[row][col] != 0) {
                        std::swap(board[target][col], board[row][col]);
                        if (target != row) board[row][col] = 0;
                        --target;
                    }
                }
            }
            break;
        case Direction::DOWN:
            // TODO
            break;
        case Direction::LEFT:
            // TODO
            break;
        case Direction::RIGHT:
            // TODO
            break;
    }
}
```

2. `merge` 合并函数

本函数实现棋盘上所有方块的合并逻辑，合并方向包括上、下、左、右四个方向。合并时，只有相邻且数值相同的方块会合并成一个新的方块，数值为原来两个方块数值之和。已完成向上合并的逻辑，你需要实现其他三个方向的合并逻辑。

```
int GameBoard::merge(Direction direction) {
    int moveScore = 0;
    switch (direction) {
        case Direction::UP:
            for (int col = 0; col < 4; ++col) {
                for (int row = 3; row > 0; --row) {
                    if (board[row][col] != 0 && board[row][col] == board[row - 1][col]) {
                        board[row][col] *= 2;
                        moveScore += board[row][col];
                        board[row - 1][col] = 0;
                    }
                }
            }
            break;
        case Direction::DOWN:
            // TODO
            break;
        case Direction::LEFT:
            // TODO
            break;
        case Direction::RIGHT:
            // TODO
            break;
    }

    return moveScore;
}
```

3. `move` 移动函数

本函数需要整合滑动和合并两个步骤，完成一次完整的移动操作。移动时，首先进行滑动操作，然后进行合并操作，最后再进行一次滑动操作，以确保所有方块都尽可能地向指定方向移动。同时需要在移动后，在棋盘上随机生成一个新的方块（2 或 4）。

```
int GameBoard::move(Direction direction) {
    // Slide and merge, anyother?
    // TODO 1

    // Add a new tile after every move
    // You can refer to the reset() function to see how to add a new tile
    // TODO 2

}
```

4. `isGameOver` 游戏结束检查函数

本函数检查当前棋盘状态是否满足游戏结束的条件。游戏结束的条件是棋盘满且无法进行任何合并。

```
bool GameBoard::isGameOver() const {
    // Check if there are any empty tiles
    // TODO

    // Check if any adjacent tiles can merge
    // TODO
}
```

5. `hasWinningTile` 获胜检查函数

本函数检查当前棋盘上是否存在数值为 2048 的方块。如果存在，则表示玩家已经获胜。

```
bool GameBoard::hasWinningTile() const {
    // Check if there is a tile with the value 2048
    // TODO
}
```

4.2 Game.cpp

1. `undoLastMove` 悔棋函数

本函数实现悔棋功能，允许玩家撤销上一步的移动操作。你需要使用栈来存储每一步的棋盘状态，以便在悔棋时恢复到上一步的状态。请注意操作顺序，根据每个人的实现不同，顺序可能有不同。

```
void Game::undoLastMove() {
    if (historyGameBoards.size() > 1) {
        // Remove current state
        // Restore previous state

    } else {
        gameBoard.reset();
    }
}
```

2. `updateGame`更新状态函数

本函数负责更新游戏状态，包括：1. 保存历史状态；2. 处理玩家的合法的move操作；3. 更新移动步数。

```
void Game::updateGame(Direction direction) {
    // Save current state to history
    // TODO 1

    // Get move score and update score
    // TODO 2

    // update move count
    // TODO 3
}
```

4.3 main.cpp

本文件主要需要你完成`while`循环以及最后的收尾工作。

4.3.1 `while`循环

1. 悔棋

```
if (input == InputType::UNDO) {
    // Implement undo functionality
    // TODO
}
```

2. 退出游戏

```
else if (input == InputType::QUIT_GAME) {
    // Think about how to handle quitting the game
    // TODO
}
```

3. 正常游戏时更新状态

```
else {
    Direction direction;
    switch (input) {
        case InputType::UP:
            direction = Direction::UP;
            break;
        ...
    }

    // Update the game state
    // TODO

}
```

4. 更新棋盘显示状态

```
// Render the updated game board
// refer to how the game is initially rendered
// TODO
```

可参考棋盘初始化的代码：

```
TerminalRender::render(game.getGameBoard(), game.getScore(), game.getMoveCount());
```

4.3.2 收尾工作

在游戏结束时，显示总计时间。

提示：可查看类`TerminalRender`相应方法。

---

5 实验运行

此处仅提供`linux`环境下的运行方法，`Windows`环境下请参考`Windows项目构建步骤说明.pdf`。

1. 首先将`zip`包解压到本地目录。

```
unzip Lab1-2048.zip -d Lab1
```

2. 进入到`CMakeLists.txt`所在目录。

```
cd <path-to-CMakeLists.txt>
```

3. 创建并进入`build`目录。

```
mkdir build
cd build
```

4. 运行`cmake`命令生成`Makefile`。

```
cmake ..
```

5. 运行`make`命令编译项目。

```
make
```

6. 运行生成的可执行文件。

```
./2048
```

注：如果缺少某些依赖包，请根据提示安装相应的依赖包。

```
sudo apt-get update
sudo apt-get install <missing-package>
```

成功编译后，终端会显示如下信息：

```
(base) bedivere@Bedivere:~/Documents/data_structure/lab1/build$ make
[ 16%] Building CXX object CMakeFiles/2048.dir/src/main.cpp.o
[ 33%] Building CXX object CMakeFiles/2048.dir/src/Game.cpp.o
[ 50%] Building CXX object CMakeFiles/2048.dir/src/GameBoard.cpp.o
[ 66%] Building CXX object CMakeFiles/2048.dir/src/InputHandler.cpp.o
[ 83%] Building CXX object CMakeFiles/2048.dir/src/GameRender.cpp.o
[100%] Linking CXX executable 2048
[100%] Built target 2048
(base) bedivere@Bedivere:~/Documents/data_structure/lab1/build$ ./2048
Welcome to 2048! Press 'E' to start game. Good luck!
+----------------------+
| Score:          0    |
| Moves:          0    |
+----------------------+

+------+------+------+------+
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
+------+------+------+------+
|    2 |      |      |      |
+------+------+------+------+
|      |      |      |    2 |
+------+------+------+------+

Available Controls:
W => Up
S => Down
A => Left
D => Right
U => Undo last move
Q => Quit game
```

否则，请检查代码是否有错误/缺少某些依赖包/步骤缺失等等。

---

6 实验分数安排与检查

6.1 分数安排

- 完成实验代码的补充，成功运行，能够完成相应功能：9分
- 现场回答相关问题：1分

注：不会涉及C++基础语法相关的问题。但可能需要你现场改一些代码以实现助教的要求。

6.2 现场检查

实验检查采用线上/线下检查两种方式。实验检查截止日期是2025年10月18日。可以截止日期之前的任意一次实验课上寻找对应助教进行实验的验收。**截止日期之后不再进行该实验的检查，未进行实验的验收的本次实验成绩登记为 0 分。**

如无特殊情况，请线下检查；否则，可与对应助教沟通进行线上检查。

实验检查时间和地点：

	时间	地点
线下	每周五 18:30-21:30	电三楼406
线上	每周六 14:00-17:00	腾讯会议