问题背景

“算24点”是一种数学游戏, 规则如下:

任意抽取 4 张扑克牌 (不含大、小王), 记下抽取的牌面上的数字, 其中 `A`、`J`、`Q`、`K` 分别视为 `1`、`11`、`12`、`13`; 目标是找到一个由这 4 个数字组成的四则运算式, 4 个数字均使用且仅使用一次, 且算式的结果等于 `24`.

问题描述

程序的核心任务是实现一个“算24点” 的计算器, 读取输入的 4 个牌面数字, 判断是否存在符合条件的算式; 如果有解, 则给出一个算式.

此外, 项目还需要实现下列额外功能:

* 文件处理功能. 程序需要读取指定文件, 文件的每一行包含一组输入; 程序分别判断对于每组输入, 是否存在符合条件的算式; 完成后, 程序可以导出一个输出文件, 内容包含每组输入和判断结果.
* 计时挑战功能. 由程序生成“算24点”题目, 用户挑战在规定时间内做出尽可能多的题目. 具体地说, 程序设置一个倒计时, 每次生成一组 4 个数字作为题目, 保证题目有解; 用户需要给出一个算式, 由程序判断它是否是符合要求的答案. 若答案正确, 用户会获得分数奖励; 若答错或放弃题目, 会相应受到扣除分数惩罚; 用户需要挑战在倒计时结束前获取高分.
* 联机对战功能. 程序可以在局域网内的两台计算机下建立连接, 两位用户分别控制一台计算机; 由程序生成“算24点”题目, 用户中最先得出正确结果的一方获得加分, 先达到指定分数的一方获胜. 由于网络相关功能有一定难度, 该功能目前暂未完成开发.

项目使用的算法、界面设计等详见下文.

测试用例

输入：`[3, 3, 8, 8]`

输出一组符合要求的解: `8 / (3 - 8 / 3) = 24`

输入：`[6, 7, 8, 8]`

没有符合要求的解, 输出: `无解`

算法设计

程序需要实现的主要算法功能包括:

* 求解“算24点”问题, 即对于给定的 4 个数字, 通过搜索回溯法, 枚举可能的运算符和运算顺序, 找到一组可行的解;
* 输出答案表达式, 即求解得到结果时, 将当前的搜索路径保存为表达式树, 并转换为中缀表达式输出;
* 表达式求值, 即处理用户挑战时输入的答案, 将中缀表达式转换为表达式树并求值, 判断答案是否满足要求.

下面列出关键的实现:

1. 有理数类 `rational` :

   考虑到浮点类型的精度问题, 定义有理数类进行程序中的四则运算. 有理数类重载了四则运算和比较运算符.

2. 表达式树模板类 `expression<T>` :

   每个对象保存一个表达式树的结点. 结点分为操作数结点和运算符结点, 操作数结点存储一个 `T` 类型的操作数对象; 运算符结点存储 2 个 `expression<T> *`, 即左右子表达式树的指针, 并以枚举类型 `enum operator_type` 的形式存储一个运算符,

   表达式树类实现了以下方法:

   (1) `T value()`, 返回表达式的值:

   对于操作数结点, 返回操作数; 对于运算符结点, 递归地求左、右子表达式的值, 根据运算符种类返回对应运算的结果.

   (2) `void print_expression_to(std::ostream &)`, 打印为中缀表达式:

   对表达式树进行中序遍历, 输出对应的操作数和运算符. 若子树的运算符的优先级较低, 应在子树的表达式两侧加括号; 若右子树的运算符的优先级相同, 但父结点的运算符为 `-` 或 `÷` 时, 打印右子树时应反转运算符.

   (3) 构造函数 `expression(const std::string &)`, 由中缀表达式建树:

   用两个栈分别保存表达式和运算符, 并维护运算符栈的单调性质. 当从字符串中读取到操作数时, 构造一个单操作数结点的表达式树, 压入表达式栈中; 当读取到运算符, 且栈顶的运算符结合优先级更高时, 栈顶的运算符出栈; 运算符出栈时, 表达式栈顶的两个表达式出栈, 它们运算得到的表达式入栈. `(` 直接进入运算符栈, 当读取到 `)` 时, 栈中运算符依次出栈, 直到与 `(` 配对.

3. 求解器类 `solver` :

   用于实现“算24点”及类似问题的求解, 以及处理求解过程中产生的临时数据.

   初始化求解器时, 需要确定操作数的数量及目标答案, 对于“算24点”问题分别是 `4` 和 `24`.

   求解器通过搜索回溯求解; 若当前有 `N` 个操作数, 程序枚举四则运算符, 并枚举从中选出 `2` 个数的所有组合 (或排列, 取决于当前运算符是否满足交换律); 对于每一种枚举情形, 将这 `2` 个数取出, 运算后放回, 得到 `N-1` 个操作数, 并调用下一层搜索, 以此类推. 最后剩下一个数, 若这个数恰好是目标答案, 判断问题有解, 此时保存对应的表达式; 若不是目标答案, 则进行回溯. 若完成了所有搜索, 没有找到目标答案, 则判定问题无解.

   文件处理允许在设置中启用多线程并发操作, 此时每个线程会创建单独的求解器实例, 确保数据互不干扰.

对于其余辅助性的实现, 可以在代码中查阅, 此处不再赘述.

界面设计

项目使用 `Qt6` 实现 GUI, 主要的界面包括:

1. 主窗口:

   左侧为菜单栏, 选中对应的菜单选项时, 右侧显示对应的界面.

2. 计算器界面:

   中部提供 4 个编辑栏, 允许用户输入“算24点”问题的 4 个参数, 并展示对应的扑克牌面;

   下方提供随机取数按钮、求解按钮和一个用于显示结果的只读编辑栏.

3. 文件处理界面:

   上方提供一个编辑栏, 用于输入待求解文件的路径; 右侧为通过浏览选取文件的按钮, 所选文件的路径也会显示在左侧编辑栏中;

   下方提供执行计算按钮、一行状态文本和保存结果按钮.

4. 计时挑战界面:

   上方为计分板, 左侧显示倒计时, 右侧为当前分数与最高记录;

   中部展示 4 个只读的编辑栏和扑克牌面, 用于展示当前题目;

   下方在游戏开始前为开始按钮; 游戏开始后, 显示输入答案的编辑栏、刷新题目按钮、提交按钮 (在编辑栏中键入回车也可以提交); 提交答案后, 编辑栏上方的文本和右侧

的图标会显示答案是否正确, 以及用户的得分变动; 游戏时间结束后, 显示开始按钮, 以及上一局的分数.

5. 设置界面:

   提供各功能所需的相关选项等.

附注

项目采用 `git` 进行版本管理, 开发进度同步到 GitHub 上; 您可以查询 `commmit` 历史记录以了解项目的开发经过;

项目使用的图标、插画等美术资源使用 `Adobe Illustrator` 绘制, 部分参考了网络上的图片资源;

作业中算法模块的设计与实现, 由学生独立完成; GUI 模块的开发, 部分借助了 AI 工具以检索 `Qt` 的技术文档和相关资料.