中国矿业大学计算机学院

23 级本科生课程设计报告

课程名称 程序设计综合实践

报告时间 2024.12.15

学生姓名 王志豪

学　　号 08232337

专　　业 数据科学与大数据技术

任课教师 张辰

《程序设计综合实践》课程评分表

序号	课程教学目标	考查方式与考查点	占比	得分
1	**目标 1：**掌握一门计算机高级语言，并能使用特定的软件开发工具，设计、开发、调试及运行应用程序。	**软件代码** 使用程序设计集成开发工具设计开发、调试应用程序，考察计算机工程基础知识	10%	
2	**目标 2：**针对具体的应用问题，进行功能需求分析，确定设计目标，并能绘制算法流程图。	**设计报告** 系统需求、系统流程图、软件界面设计、关键类图及软件扩展描述，考察问题分析能力	40%	
3	**目标 3：**在进行需求分析的基础上，设计软件运行界面、关键类、编写代码，调试并正确运行满足需求的应用程序。	**软件代码** 软件代码编写、调试、运行演示、系统功能扩展，考察计算机工程实践能力	50%	
总分			100%	

评阅人：

《程序设计综合实践》课程指导教师评语

指导教师签名	

目　录

实验一 简单计算器

1.1 系统概述

本项目目标是设计开发一个支持连续计算的四则运算计算器 myCalculator，通过单击按钮，输入并完成如 4+5+6 或 5*8+16 类似的连续计算，并将运算结果显示再输出文本框中，同时也具备清空、后退等功能。在完成以上基本功能的基础上，本项目还额外添加了例如求余运算、求根号运算、求次方运算以及取倒数等进阶运算，而且本项目还在主页面中嵌入了一个 BMI 计算器，用以计算人体 BMI 指数。系统采用 Python 语言（3.12.6 64-bit）进行开发，编辑器为 Visual Studio Code。

1.2 系统设计

1.2.1 设计目标

该项目设计目标是开发一个支持连续计算并且保证运算优先级的四则运算计算器，通过单击按钮输入并完成计算，将结果显示在输出文本框中。同时有清空和退格、浮点运算、括号功能。项目扩展部分是身高体重 BMI 计算器以及求余运算、求根号运算、求次方运算以及取倒数等进阶运算。

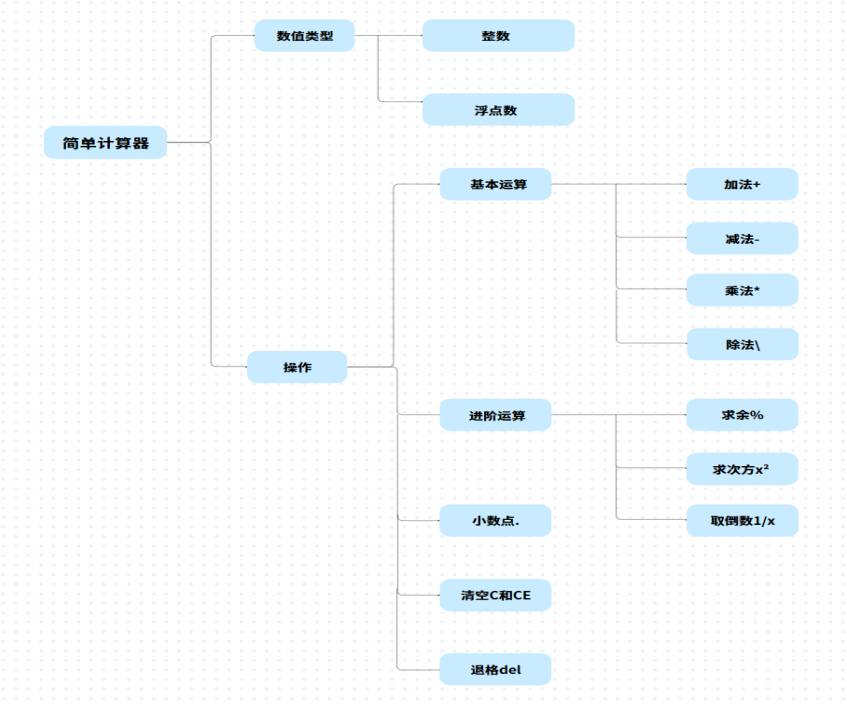


图 1-1 计算器设计目标

1.2.2 设计分析与算法流程

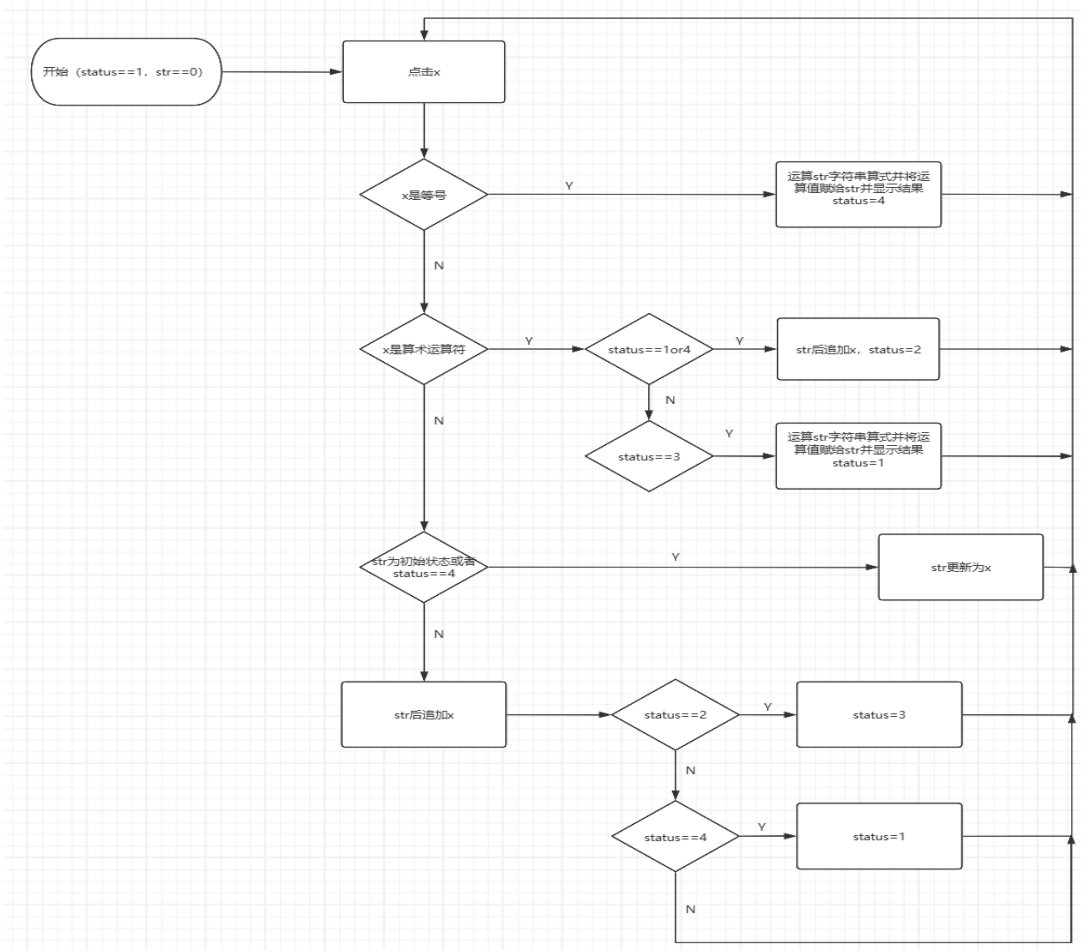


图 1-2 算法设计流程图

设计分析

如图所示，本项目的算法的主要逻辑是基于 Windows 系统计算器功能所设计的。本项目的算法逻辑主要分为两个方面，第一是**控制运算状态（图中 status 便是运算状态的参量，status==1 表示现在正在输入第一个数，status==2 表示现在输入的是运算符，status==3 表示现在输入的是第二个数，status==4 表示按下等于号后执行运算**），第二是**在输入过程中保存运算符左边的数（第一个数）和运算符右边的数（第二个数），调用 Python 中的运算模块，将这两个保存的数进行相应的运算。**

对于第二点来说，具体的程序中 calcMainWindow 类中具有成员 num_1、num_2 和_operator（_operator 是接收 Python 中运算模块的函数的，也就是说，在点击运算符按钮时，该_operator 会被赋成一个函数）

具体逻辑如下：

当开始时，str==0，status==1，当输入数字时，此时要将 0 置换为相应输入的数，在输入相应的运算符的时候，程序会保存 str 中的数字，成员 num_1 就是保存第一个数，此时_operator 被赋为对于输入运算符的函数，status=2；后来再输入数字后 status=3，相应的第二个数被保存在 num_2 中。若此时不点击“=”而是再次点击运算符，此时也会调用 _operator 函数将 num_1 和 num_2 传入后运算返回一个结果，status=1 不过该结果会被更新为 num_1，因为还会进行后续的运算，这样就会构成一个循环；若直接点击“=”，则调用_operator 函数将 num_1 和 num_2 传入到 _operator 函数中运算并返回结果，status=4 循环结束；那么我要进行下一次运算的时候，status 又会被重新赋为 1，表示循环重新开始。

1.2.3 界面设计

图 1-3 计算器主界面设计

1.2.4 关键类图

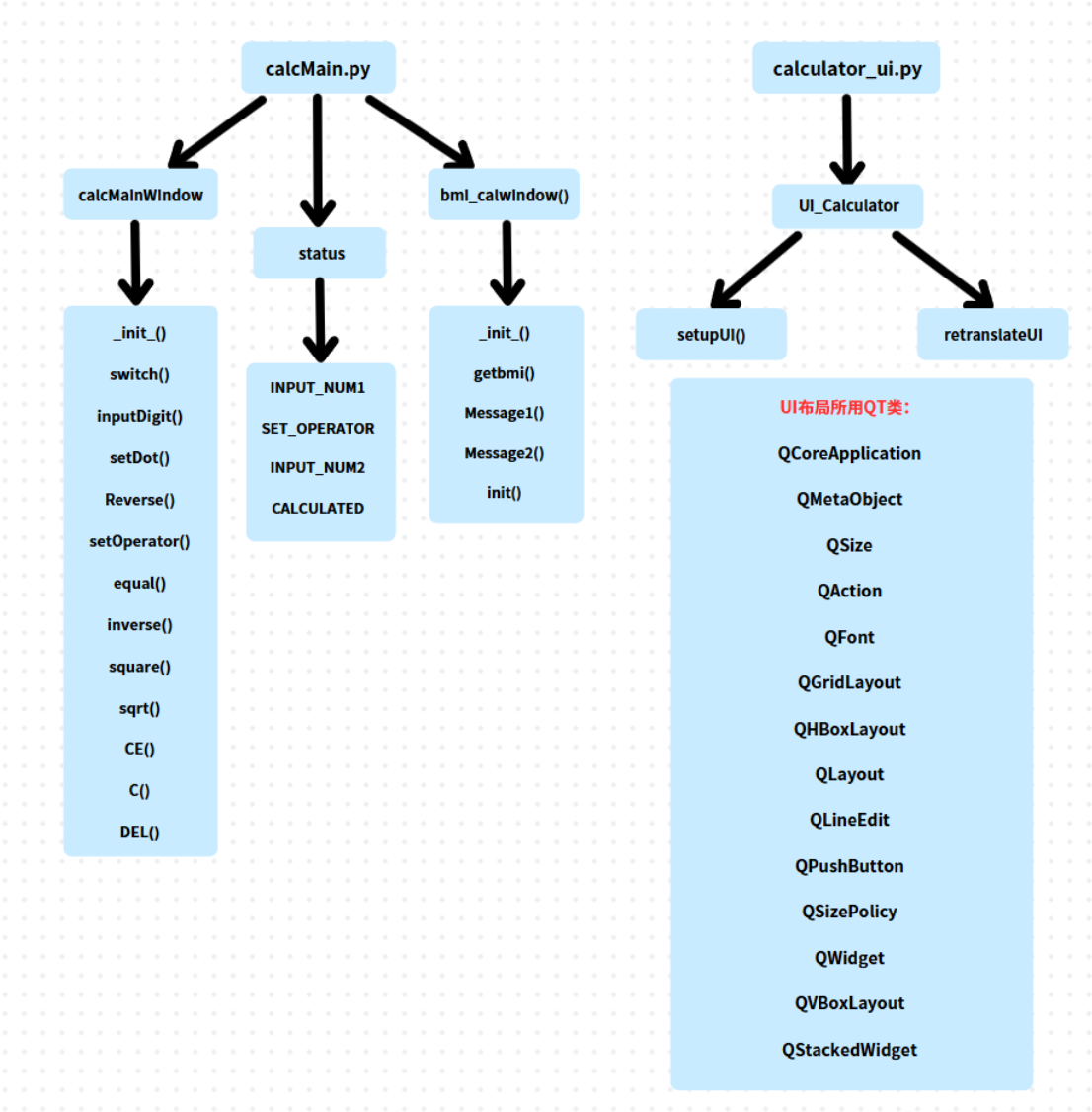


图 1-4 关键类图及函数

UI 设计所涉及的 QT 类均为在使用 QT Designer 进行完图形化编程后转译的 Py 文件中为各按键或者显示框单元所赋的类，例如在本项目中数字按键 1-0 分别为 oneButton——zeroButton，它们均为 QPushButton 类，再例如结果显示框 digitDisplay 就是 QLineEdit 类。具体命名详见 bmi_cal_ui.py 和 calculator_ui.py 文件。

1.3 系统实现（运行调试）

1.3.1 加减乘除基本连续运算

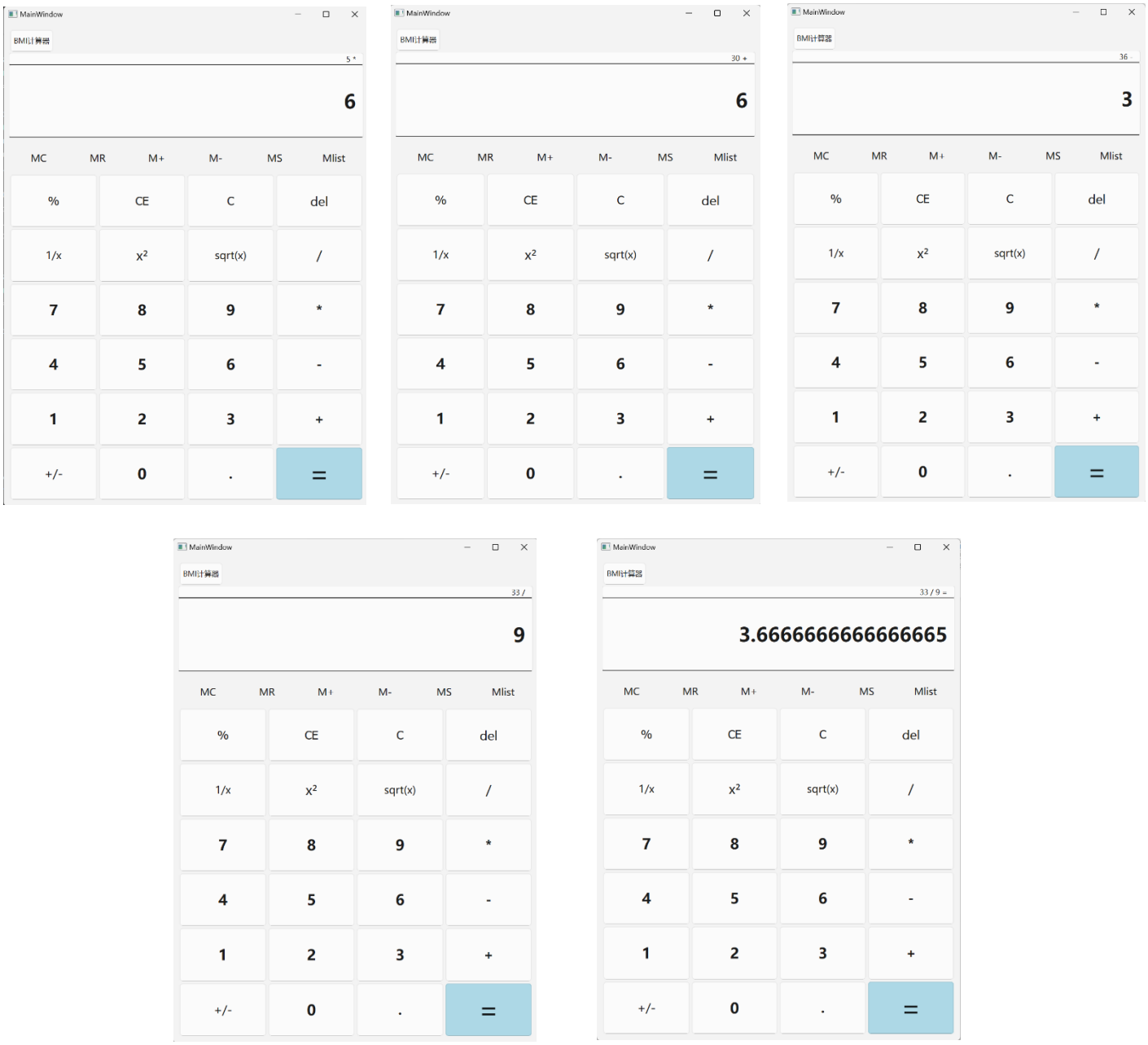


图 1-5 加减乘除基本连续运算

如上图，参照 Windows11 系统的计算器，设计出的计算器具有如上连续计算的功能，此功能具体表现在执行下一次运算符运算的时候，会先将之前的运算完成，并输出结果，再完成后续的运算。

1.3.2 退格、清空、加小数点功能

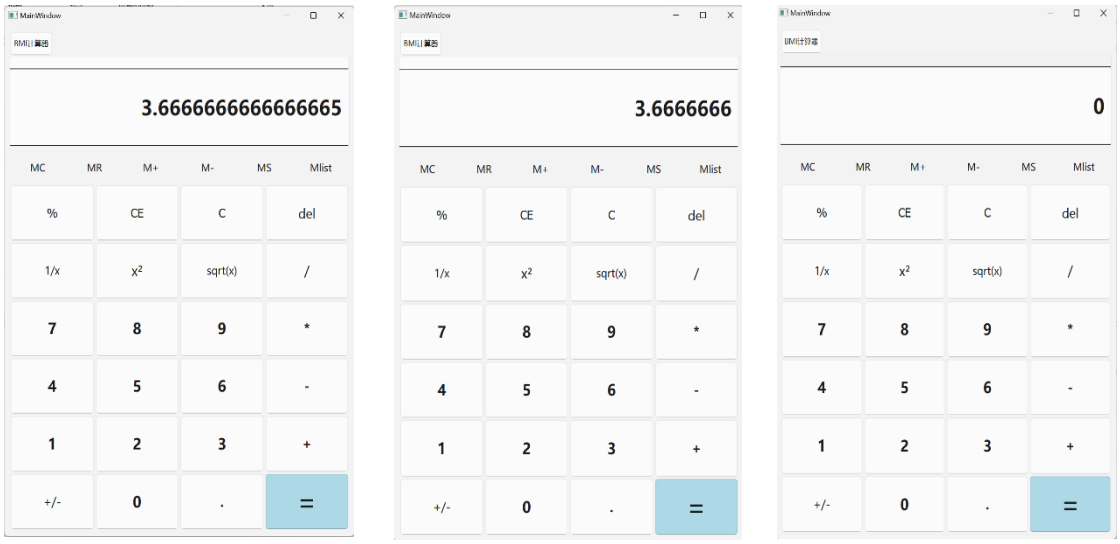


图 1-6 退格、清空、加小数点操作

如上图所示，先输入“3.6666666666666665”，在输入过程中，点击小数点可添加小数点；连续点击 9 次“del”键可实现 9 次退格操作；而后点击“C”键，便可实现清零操作。

1.4 系统扩展

1.4.1 BMI 指数计算器

标准体重是反映和衡量一个人健康状况的重要标志之一。过胖和过瘦都不利于健康，也不会给人以健美感。不同体型的大量统计材料表明，反映正常体重较理想和简单的指标，可用身高体重的关系来表示。BMI 指数（英文为 Body Mass Index，简称 BMI），是用体重千克数除以身高米数的平方得出的数字，是国际上常用的衡量人体胖瘦程度以及是否健康的一个标准。当我们需要比较及分析一个人体的体重对于不同高度的人所带来的健康影响时，BMI 值是一个中立而可靠的指标。本程序设置了 2 个 QLineEdit，当用户输入身高和体重的时候，可以计算出用户的 BMI。

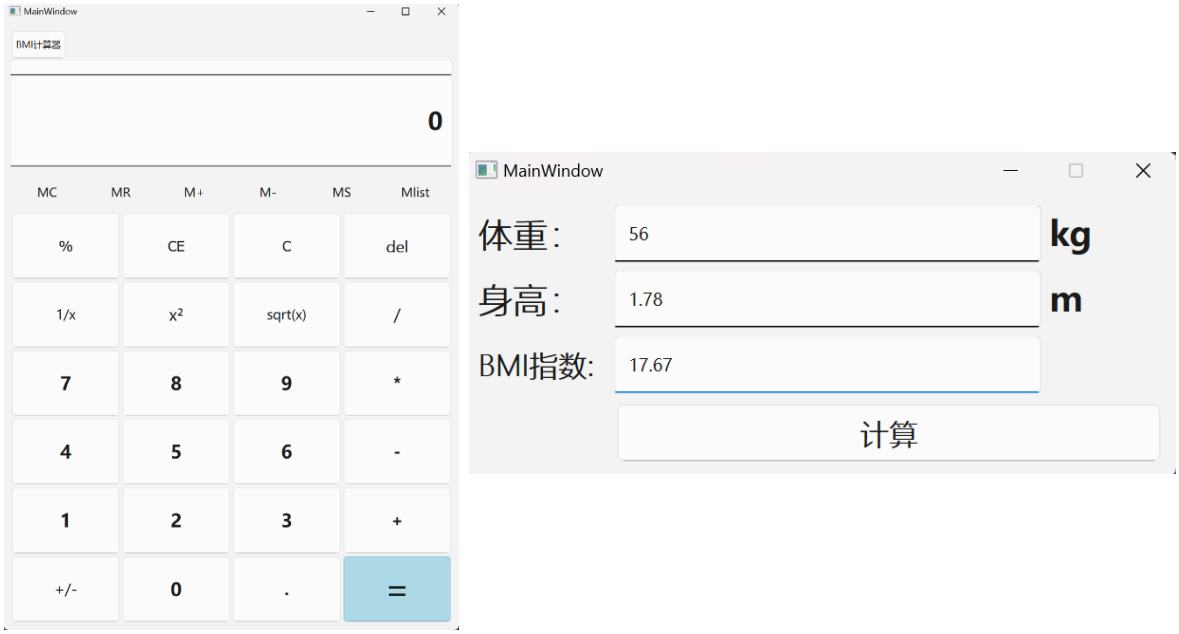


图 1-7 BMI 计算器

如上图所示，点击主页面左上角的“BMI 计算器”按钮便可调出 BMI 计算器界面，在其中输入对应的体重和身高数据，点击“计算”按钮便可计算出对应的 BMI 指数值。若体重和身高的数据栏中其中一个没有输入数据都会报出“数据类型错误”的提示，提醒重新输入数据。

1.4.2 取倒数、求余、求次方、求平方根

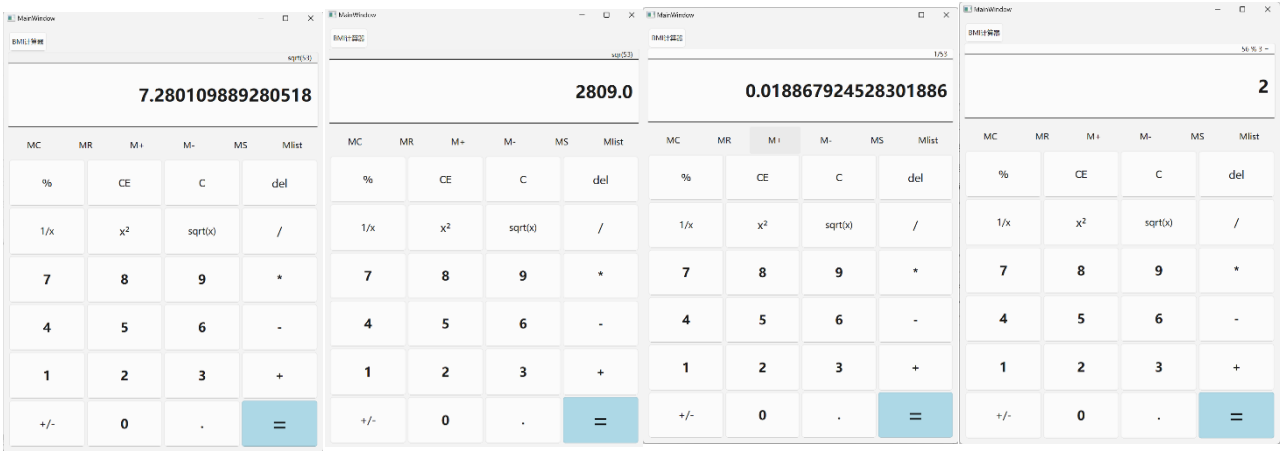


图 1-8 计算器主界面设计

如上图所示，从左至右分别是求平方根、求次方、取倒数、求余的运算。

1.5 总结

本项目主要实现了计算器的基础功能，并在实现基础功能的基础之上加入若干扩展功能，如求余等运算以及 BMI 计算器，在开发的过程中以 Windows11 系统的计算器为功能蓝本，完成了基本的功能复刻。通过该项目的开发，我首先对 Python 语言更为熟悉，并且初步掌握了利用 Qt Designer 开发人机交互界面的整体过程和方法，以及如何在设计计算器逻辑的过程中调用这些按钮类。今后对 UI 设计的美观度还需更加注意。

实验二 拼图游戏

2.1 系统概述

本项目目标是设计开发一个支持鼠标拖动拼图的游戏软件 myPuzzle，软件能够自动对加载的图片进行分割，并打乱顺序后放置在不同的图片匡中，用户使用鼠标拖动图片框中的图片进行拼图，拼图成功后，系统会自动进行提示。此外本项目还基于基本功能，增添了调整难度、计时、挑战模式。

2.2 系统设计

2.2.1 设计目标

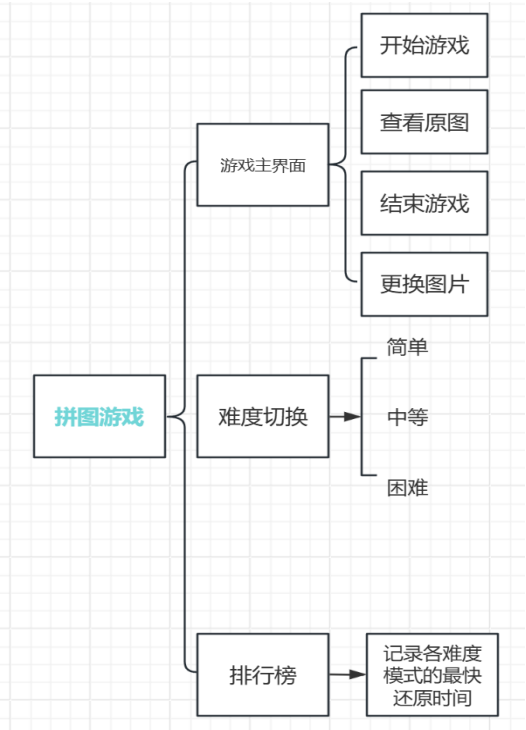


图 2-1 拼图游戏设计概述

图片尺寸自适应：为方便用户能够对不同尺寸的图片进行拼图，软件加入了图片尺寸自动调整功能，能对不同尺寸的图片进行自动调整以满足拼图游戏的要求。

图片动态分割：能自动将图片分割成不同的大小，如 3×3 矩阵或其他大小的矩阵，便于控制拼图游戏的难易程度。

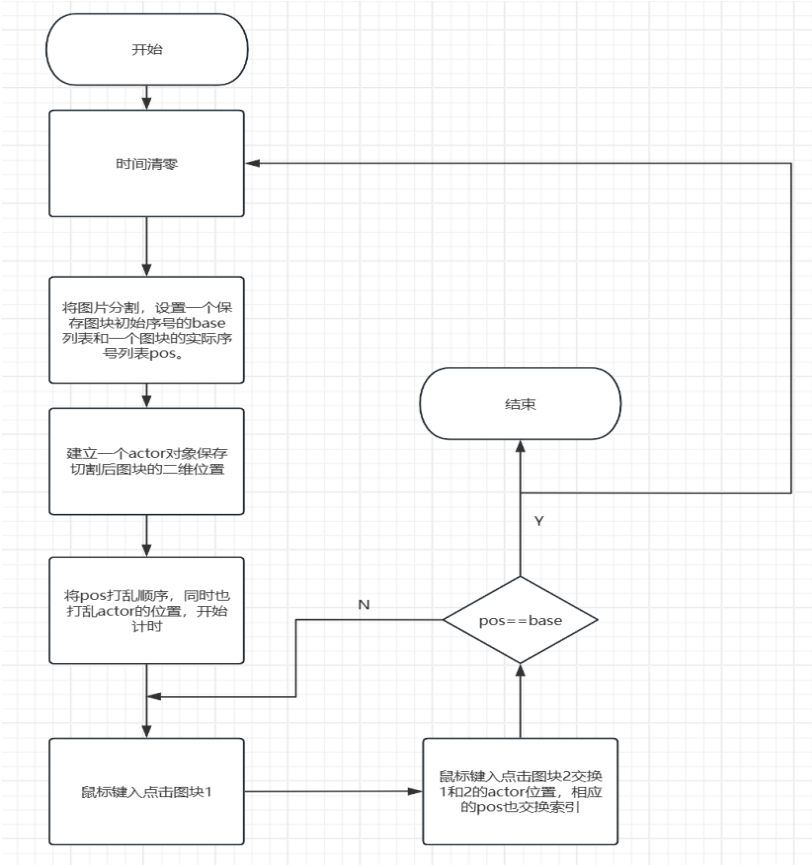
查看原图：为方便拼图，允许用户在拼图过程中随时查看原图。

随机切换图片：能在软件自带图片中进行随机切换，也可以另外选择用户自己喜欢的图片进行拼图，增加趣味性

鼠标拖动图片：选择相应图片框中图片，按下鼠标左键，再选中另一张图片即可完成两图片位置的交换，以达到拖动图片的效果。

自动判断拼图成功：软件自动记忆图片切割后的原始状态，能够对用户拼图是否成功自动做出判断。

2.2.2 设计分析与算法流程



如上图所示，其中所体现的是本游戏的基本核心逻辑，即通过将二维位置交换转化为一维的位置交换，最后在判断游戏是否结束时，用的也是比较经过交换后的列表与原列表是否完全相同来判断的。其二维坐标转换成一维坐标可以实现类似函数式的一一对应关系

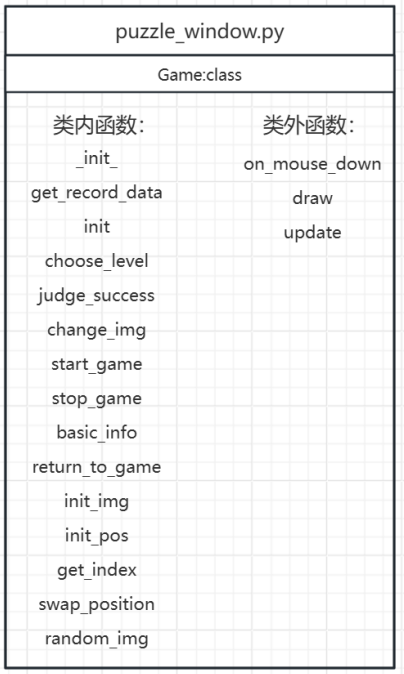
```
def get_index(self, i, j):
    return int(self.split_n * i + j)
```

如上图所示，一维坐标 index=步长×横坐标 i+纵坐标 j。

2.2.3 界面设计


myPuzzle
游戏尚未开始
请选择挑战难度：
简单
中等
困难
开始游戏
结束游戏
查看原图
更换图片
榜单记录：
简单：暂无记录
中等：暂无记录
困难：暂无记录

2.2.4 关键类图



2.3 系统实现（运行调试）

2.3.1 基本功能

1．点击开始游戏后，图片被自动切割并且切割后的图片随机分布，此时计时器开始计时

2．点击查看原图按键，可以在游戏进行中查看原图，按下返回拼图便可继续游戏

3．点击更换图片和结束游戏都会返回主页面，重新开始游戏，此时计时器会清零，游戏重新开始。

4.在完成拼图后，软件会提示“恭喜完成游戏”并且自动结束计时。并且显示还原所用时间

2.4 系统扩展

2.4.1 挑战难度选择

本游戏可供玩家选择三种游戏难度，分别为“简单”、“中等”、“困难”，它们分别对应了 3×3、4×4、5×5 的拼图分割方式，在游戏开始前，玩家可通过难度模式选择自己喜欢的难度进行挑战。

2.4.2 最佳记录榜单：

本游戏设置了一个记录榜单，用于记录玩家进行游戏时在各个难度模式下的最佳时间记录，该记录会持续保存，直到新的记录出现会更新原有的记录。

2.5 总结

通过本次拼图游戏的设计，我对 Python 中的 Pygame 库有了深刻细致的学习，在不断学习库中的形形色色的函数的过程中，我开始思考一款游戏的基本要素应当是什么，对于一款游戏，开发者视角和玩家视角之间到底有什么差别？在独立搭建完一款游戏后，我对这些问题有了深刻的认识。此外，因为拼图游戏的需要，我还学习了图像处理相关的知识，其中的核心便是 PIL 库，但是对于拼图而言，简单的调用库函数并不能完成拼图的要求，面对这样的问题，我基于 PIL 库思考了如何将切割的图像进行随机分布，通过不断试错，最终完成了游戏的开发。

实验三 多文档文本编辑器

3.1 系统概述

该项目目标是设计开发一个简单的多文档文本编辑器 **SimpleMDIExample**，具有新建、打开、保存一个文本文件；设置字体、字型功能。本项目基于 Python 语言（3.12.6 64-bit）以及 Qt designer 进行开发。

3.2 系统设计

3.2.1 设计目标

- ✓ 打开已存在文档：可以打开以.txt 结尾的文本文档，读取文件内容并将其显示在文本框中。
- ✓ 新建文档：新建一个子窗体，其文本编辑框处于空白状态。
- ✓ 保存文档：将文本编辑框中的文本保存到.txt 文件中。
- ✓ 设置字体：对选中的文本设置字体。
- ✓ 设置粗体：选中的文本粗体显示。
- ✓ 设置斜体：选中的文本斜体显示。
- ✓ 设置下划线：对选中的文本设置下划线。
- ✓ 设置窗体排列方式：对子窗体设置排列方式

3.2.2 设计分析与算法流程

由于本项目的基本逻辑实现依赖于 Pyqt5 中所提供的功能类，所以设计该项目的核心在于 UI。

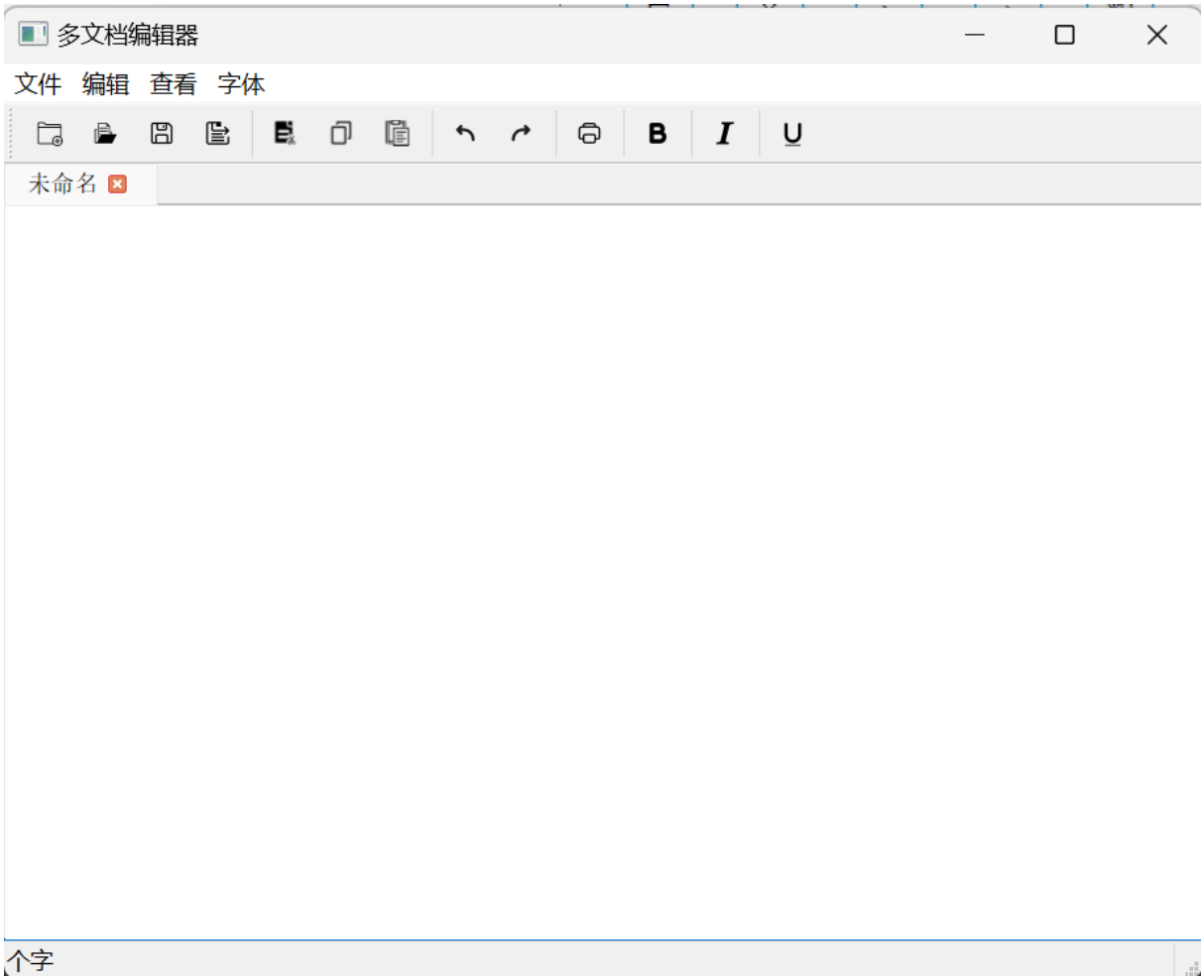
新建：在程序中，我创建了一个 QTabWidget 控件专门用来存放新建的文档，每点击一次新建，程序会创建一个 QTextEdit 型的变量存入这个控件中。

保存：将文本编辑框中的文本保存到.txt 文件中，具体通过调用 QFileDialog 中的 getSaveFileName 实现。

打开：程序会创建一个专门存储打开文件位置的 path 变量，该变量会接收选中的文件位置，并通过一个 text 变量接收该文件的名称，然后用新建过程中的 add_new_tab 函数进行添加文件进入 QTabWidget 控件中。

设置字体、加粗、下划线、斜体和颜色：分别调用 QTextEditl 类中的 setCurrentFont 、 setFontWeight 、 setFontUnderline 、 setFontItalic 、 setTextColor 函数。

3.2.3 界面设计

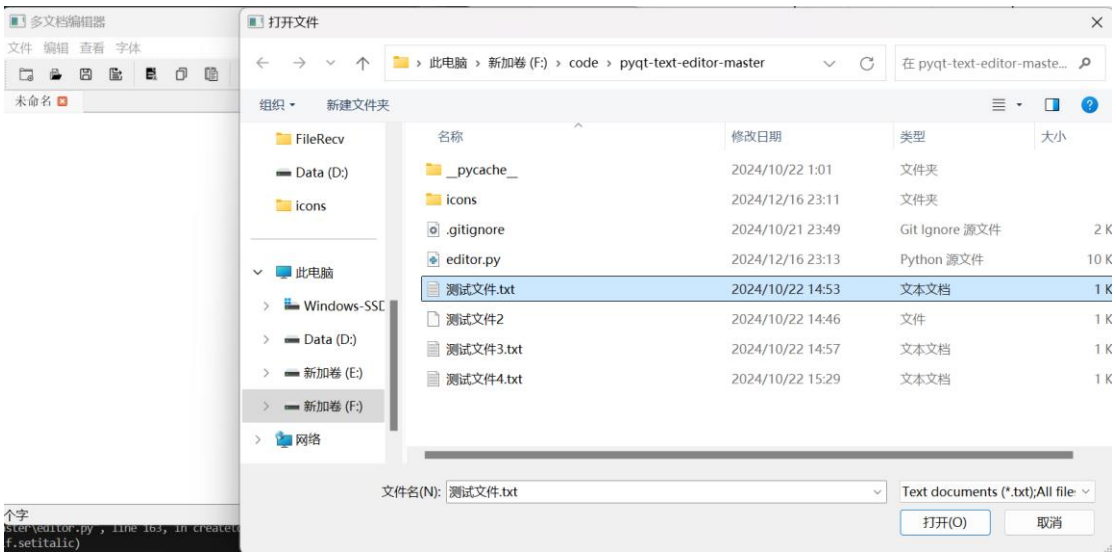


界面主要由五部分组成，分别是顶层的菜单项层，第二层的功能选项层，第三层的文件管理层，中央的文本编辑区域，最底部的计数区域。

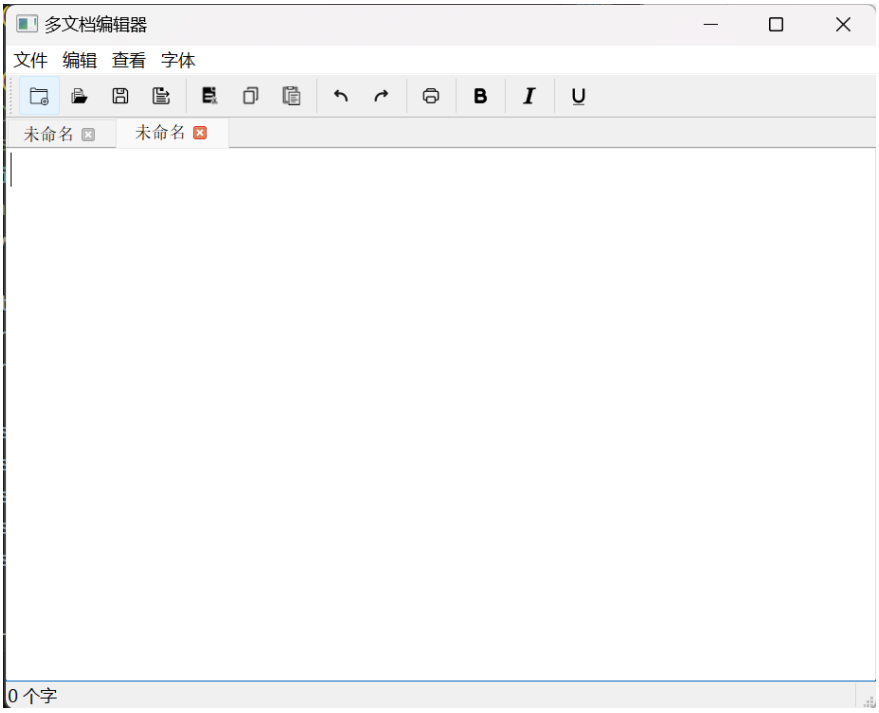
3.2.4 关键类图

3.3 系统实现

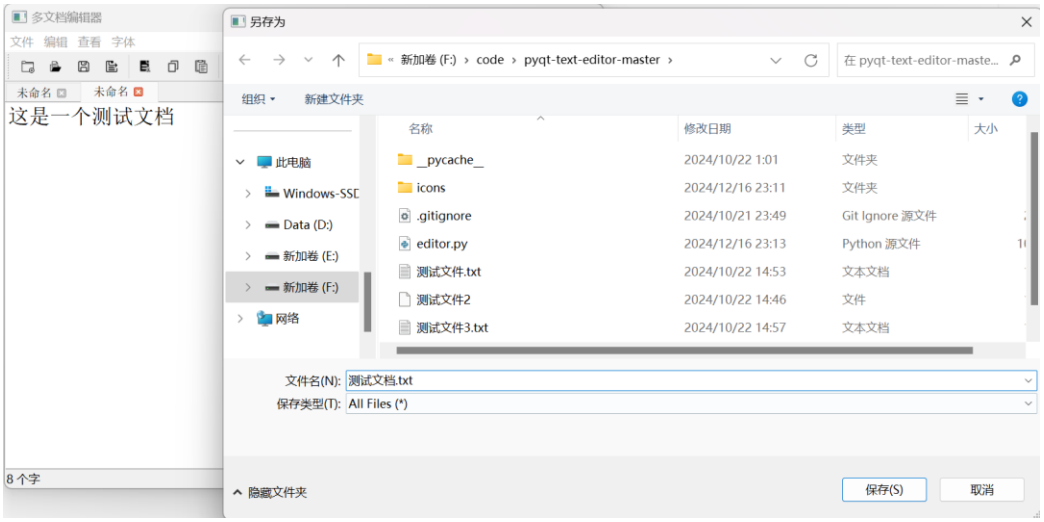
3.3.1 基本功能

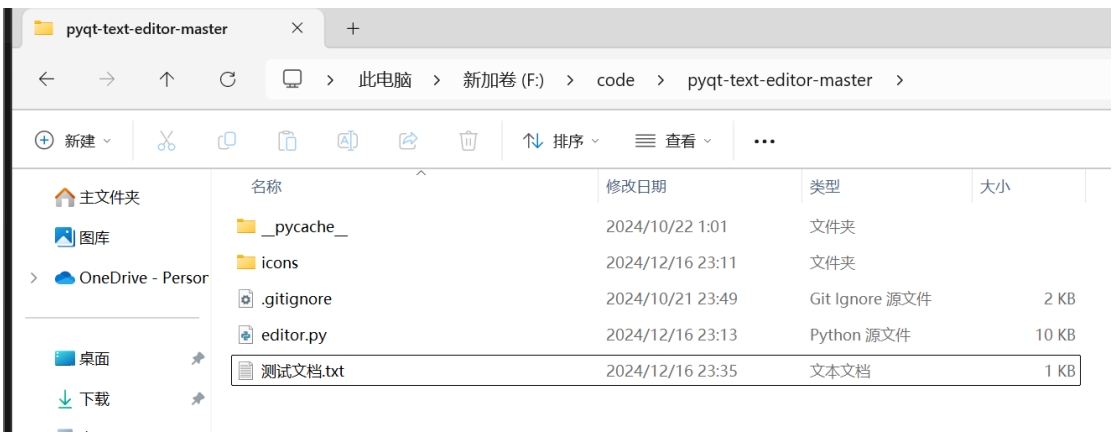


1. 点击左上角第二个“打开”图标，可以选择文件打开

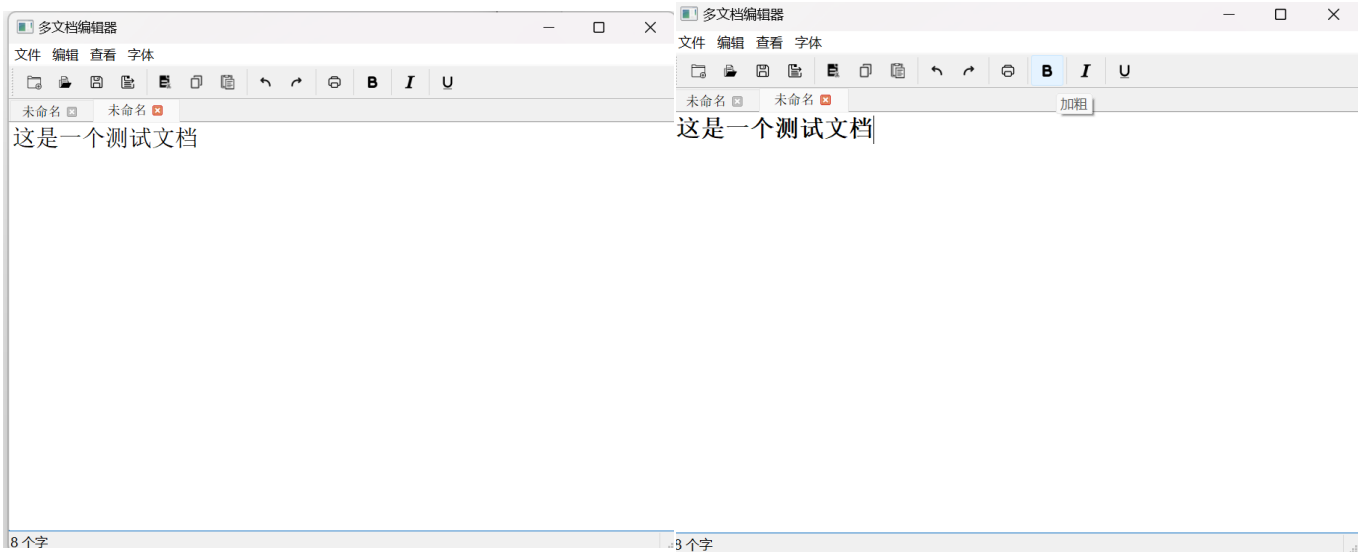


2. 点击左上角第一个“新建”图标，可以新建一个 txt 文件

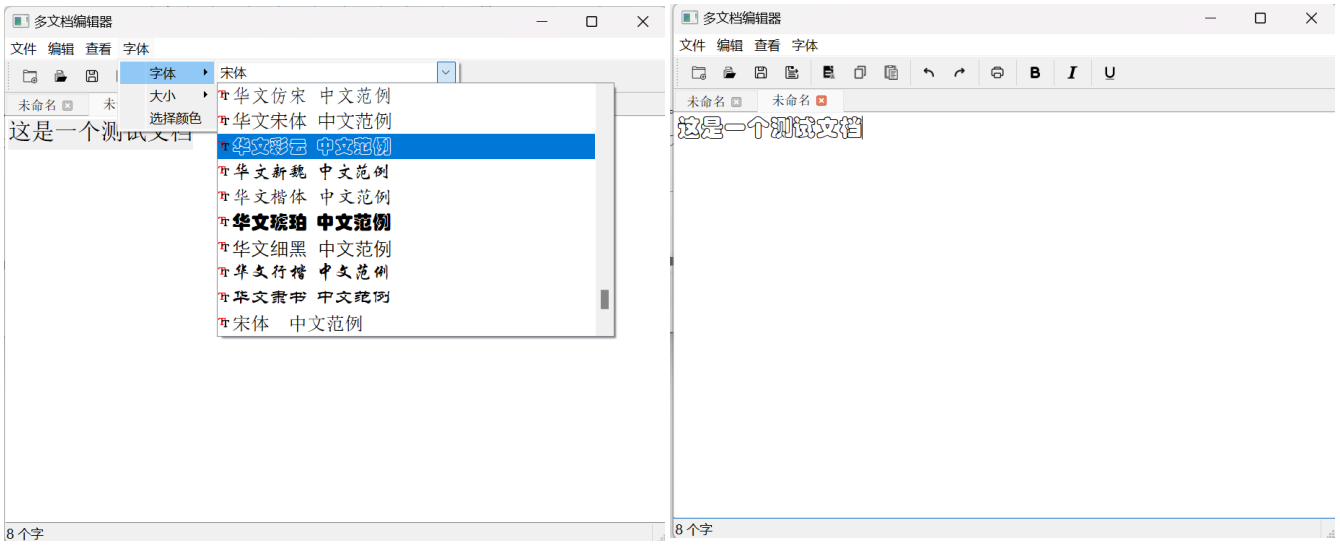




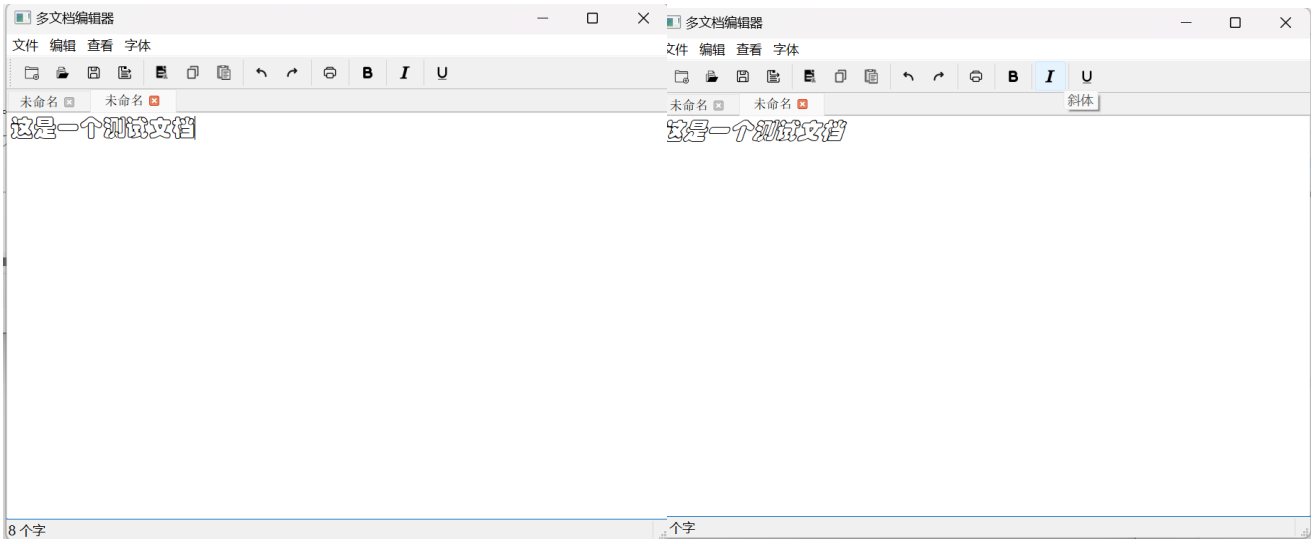
3. 点击左上角第四个“另存为”按钮即可另存为文件



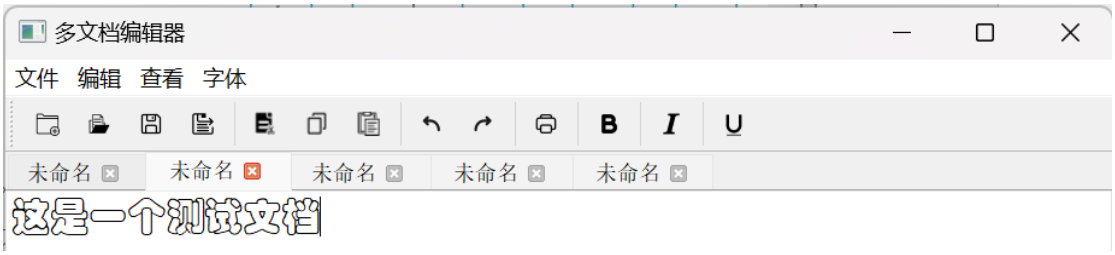
4. 点击“加粗”按钮即可设置字体加粗



5. 点击字体选项，选择想要的字体，即可完成改变字体



6. 点击“斜体”按钮即可使文字变成斜体

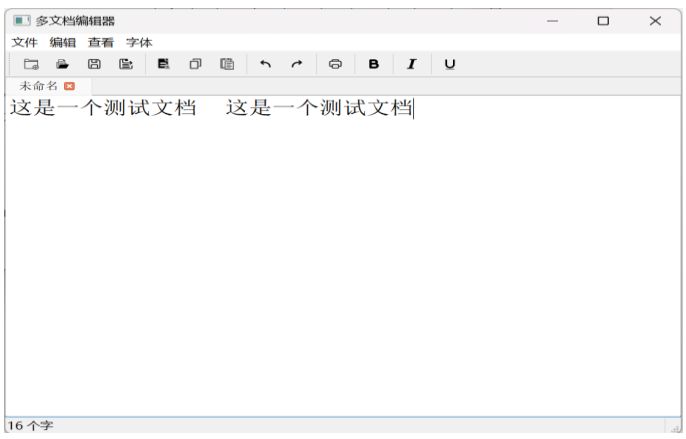


7. 如图所示，本项目创建了一个状态栏，该状态栏中可容纳若干文件，实现了窗体的有序水平排列。

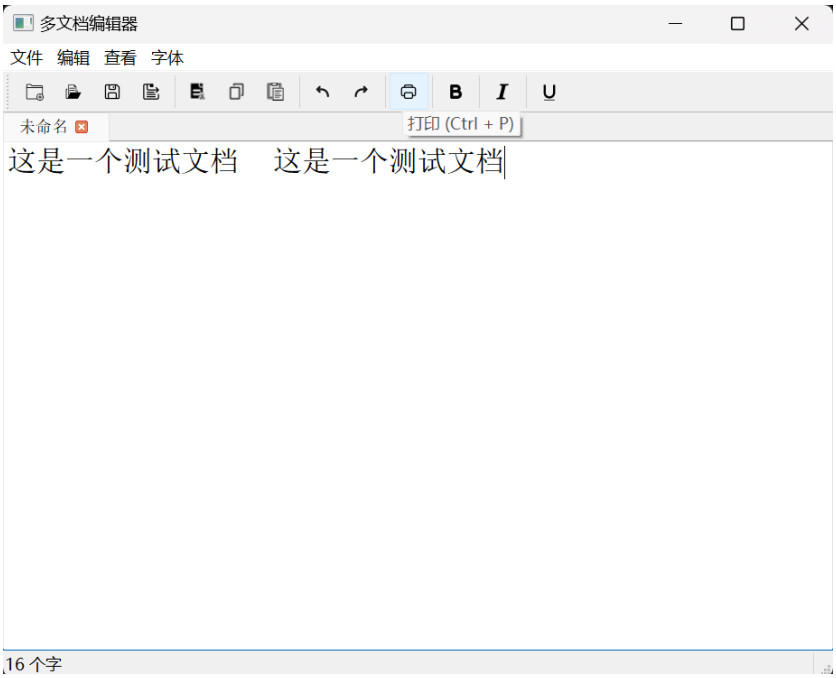
3.4 系统拓展

3.4.1 记录字数

如图所示，本文档编辑器可以实现字数的有效记录，即不将空格记录进文字总数中。



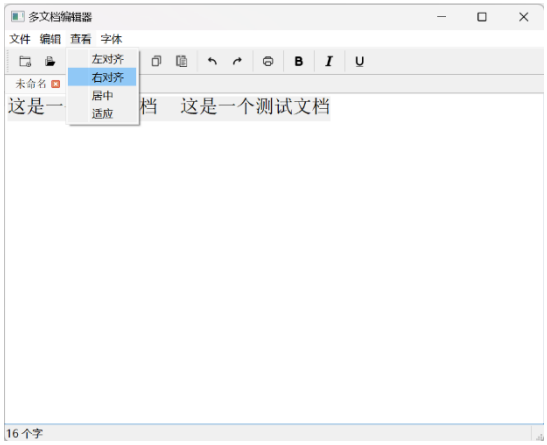
3.4.2 连接打印机

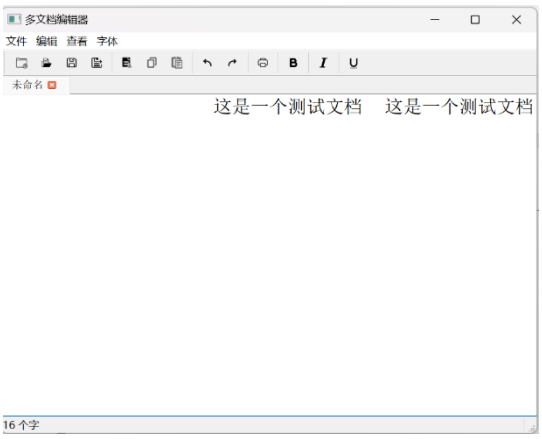


如上图所示，在点击“打印机”按钮后，系统的打印程序便被自动呼出，在连接打印机的情况下可以将编辑的文档送入打印机中进行打印。

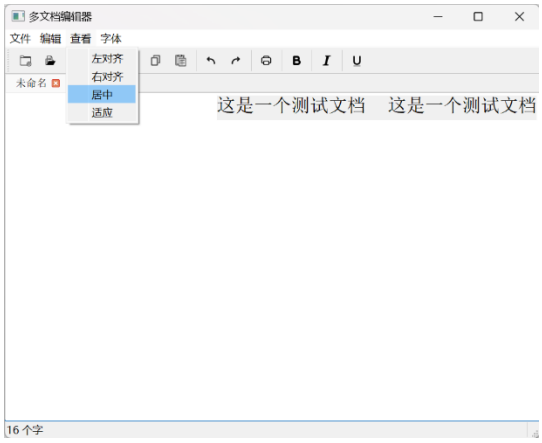
3.4.3 文字的对齐

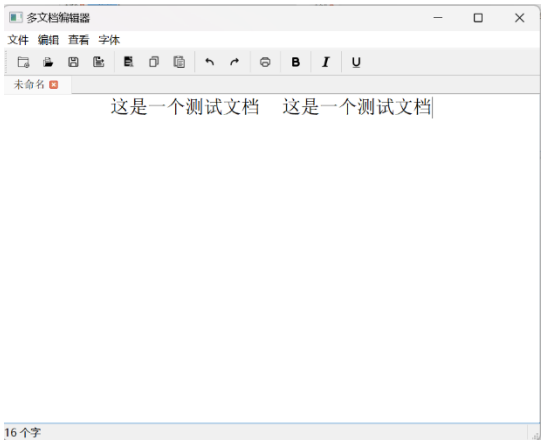
本文本编辑器还可以实现文本的对齐功能，其中包括，左右对齐、居中以及适应功能。如下图所示演示的是右对齐和居中操作，左对齐和适应与之类似。





点击查看中的“右对齐”按钮便可实现右对齐。





点击查看中的“居中”按钮便可实现居中。

3.5 总结

在设计文本编辑器的过程当中，我对 Qt 开发有了更加深入的学习，虽然文本编辑器在逻辑上实现并不困难，但是在布局的美观程度上挑战不小，而且文本编辑器的功能多，在实现每个功能的过程中也得考虑其他功能是否会产生冲突，在开发完成编辑器后，我对这些问题都有了新的认识，这样也提醒我在今后的开发过程中，要注意人机交互界面的美观和“人性化”，要在实现功能的基础上对产品进行优化。

实验四 CSP 模拟认证考试训练

4.1 题一

4.1.1 题目描述

密码

时间限制： 1.0 秒

空间限制： 512 MiB

相关文件： 题目目录（样例文件）

刷新

题目背景

西西艾弗网对用户密码有一套安全级别评定标准。

题目描述

在西西艾弗网上，用户的密码是一个由大写字母（`A-Z`）、小写字母（`a-z`）、数字（`0-9`）和特殊字符（`*` 和 `#`）共 64 种字符组成的字符串。

根据复杂程度不同，密码安全度被分为高、中、低三档。

- 高：由上述 64 种字符组成，长度大于等于 6 个字符，包含字母、数字和特殊字符，同一个字符出现不超过 2 次；
- 中：由上述 64 种字符组成，长度大于等于 6 个字符，包含字母、数字和特殊字符，且未达到高安全度要求；
- 低：由上述 64 种字符组成，长度大于等于 6 个字符，且未达到中安全度要求；

小 P 为自己准备了 n 个候选密码，试编写程序帮小 P 自动判别每个密码的安全级别。保证这 n 个密码都至少满足低安全度要求，当安全度为高、中、低时分别输出 `2`、`1`、`0` 即可。

输入格式

从标准输入读入数据。

输入共 $n+1$ 行。

第一行包含一个正整数 n，表示待判别的密码个数；

接下来 n 行，每行一个字符串，表示一个安全度至少为低的候选密码。

输出格式

输出到标准输出。

输出共 n 行，每行输出一个整数 `2`、`1` 或 `0`，表示对应密码的安全度。

4.1.2 算法分析：

```
int result(string a){
    int length=a.length();
    int flag1=0;
    int flag2=0;
    int flag3=0;
    for(int i=0;i<length;i++){
        if((a[i]<='z'&&a[i]>='a')||(a[i]>='A'&&a[i]<='Z')){
            flag1=1;
            continue;
        }
        if(a[i]>='0'&&a[i]<='9'){
            flag2=1;
            continue;
        }
        if(a[i]=='*'||a[i]=='#'){
            flag3=1;
            continue;
        }
    }
    int count=1;
    int flag=0;
    if(flag1&&flag2&&flag3){
        for(int i=0;i<length;i++){
            count=1;
            for(int j=i+1;j<length;j++){
                if(a[i]==a[j])  count++;
            }
            if(count>2) return 1;
        }
        return 2;
    }
    else    return 0;
}
```

本题我定义了一个函数 result（string a）:int，该函数的作用是判断字符串是属于哪个安全级别并返回安全级别代号。

对 result 函数进行具体分析：我设置了三面旗帜，flag1、flag2、flag3 初始值均为 0，先对字符串进行遍历，找到了字母，则 flag1=1；若找到了数字，则 flag2=1；若找到了特殊字符，则 flag3=1。遍历完成后，对于低安全的字

符串，这三面旗帜必有至少一面为 0，则直接返回 0；对于这三面旗帜均为 1 的字符串，说明该字符串至少是中安全度，现只要重新遍历字符串，判断是否有重复两次以上的元素出现即可，定义一个计数 count，用两层循环逐个遍历，内层循环每结束一次就比较一次 count 和 2 的大小，出现 count>2 时直接返回 1，表示中安全度；若外层循环结束了还没有返回值，则返回 2 表示高安全度。

125776	王志豪	密码	g++	Accepted	100	2024-11-30 15:48:16

4.2 题二

4.2.1 题目描述

字符串变换

时间限制： 1.0 秒　　刷新

空间限制： 512 MiB

相关文件： 题目目录（样例文件）

题目描述

本题涉及字符包括大小写字母（`A-Z` 和 `a-z`）、数字（`0-9`）和空格共 63 种。在这个字符集合上，小 P 定义了一个字符替换函数 $f(ch)$，表示将字符 ch 替换为 $f(ch)$。例如 $f(a)=b$ 表示将 `a` 替换为 `b`，$f(b)=0$ 表示将 `b` 替换为 `0`。进而可以将其扩展为字符串变换函数 $F(s)$，表示对字符串 `s` 进行变换，将 `s` 中**每个**字符 ch 都替换为 $f(ch)$。

字符替换函数 f 可表示为 n 个字符对 (ch_1,ch_2)，即 $f(ch_1)=ch_2$。

- n 个字符对中，ch_1 两两不同，即不会出现同时定义了 $f(a)=b$ 和 $f(a)=0$ 的情况；
- 未定义的 $f(ch)$，可视为 $f(ch)=ch$，即字符 ch 保持不变；
- 函数 f 为单射，即当 $ch1\neq ch2$ 时有 $f(ch1)\neq f(ch2)$，例如不会同时有 $f(b)=0$ 和 $f(0)=0$（`b` 和 `0` 都被替换为 `0`）。

现给定初始字符串 `s`，试处理 m 个查询：每个查询包含一个正整数 k，询问对初始字符串 `s` 变换 k 次后的结果 $F^k(s)$。

输入格式

从标准输入读入数据。

输入共 $n+4$ 行。

输入的第一行包含一个字符串，形如 `#s#`，即用两个井号字符 `#` 将初始字符串 `s` 囊括其中。

输入的第二行包含一个正整数 n，表示组成函数 f 的字符对数；接下来 n 行每行输入一个形如 `#xy#` 的字符串，表示 $f(x)=y$。

输入的第 $n+3$ 行包含一个正整数 m，表示查询的个数；下一行包含空格分隔的 m 个正整数 $k_1,k_2,\cdots,k_m$，表示 m 个查询。

输出格式

输出到标准输出。

输出共 m 行，依次输出 m 个查询的结果；输出时每行同样是一个形如 `#s#` 的字符串，即用两个井号把变换后的字符串 `s` 括起。

4.2.2 算法分析

```
void change(int k){
    string c=a;
    for(int i=0;i<k;i++){
        for(int j=0;j<c.length();j++){
            for(int z=0;z<n;z++){
                if(c[j]==b[z][1]){
                    c[j]=b[z][2];
                    break;
                }
            }
        }
    }
    for(int l=0;l<c.length();l++){
        cout<<c[l];
    }
}
```

在最初分析时，我采用的是最直接的暴力枚举，就是把每次需要变换的 k 次直接传入函数中，然后进行三次循环遍历，这样会导致时间复杂度非常大。正如下图所示，最后评测出的结果只有 80 分，时间超限。

125783	王志豪	字符串变换	g++	Time Limit Exceeded	80	2024-11-30 15:50:01

随后我对算法进行了优化，引入了循环周期的量度 zhouqi[ch]。其实现的方法具体如下：

遍历字符串中的每个字符 ch。

若 ch 存在循环周期 zhouqi[ch]：

映射次数 k 可以通过 k % zhouqi[ch] 优化为最多遍历一次完整循环。

遍历 k % zhouqi[ch] 次，找到最终的字符。

若 ch 没有循环周期（即它只映射到自身），直接返回 ch。

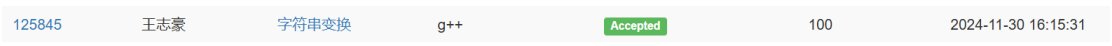
```
string getResult(int k) {
    string ret;
    for (const auto& ch : s) {
        char newChar = ch;
        int k1 = k;
        if (zhouqi[ch] != -1)
            k1 %= zhouqi[ch];

        for (int i = 0; i < k1; i++) {
            if (newChar == f[newChar])
                zhouqi[newChar] = i + 1;
            else
                newChar = f[newChar];
        }
        ret.push_back(newChar);
    }
    return ret;
}
```

对于每个字符 ch：

1. 如果当前 zhouqi[ch] 是 -1，说明还未确定其循环周期，进入计算。
2. 通过从 ch 开始沿着映射链表，记录经过的字符，直到回到起点 ch。
3. 循环的长度（访问的字符数）即为 zhouqi[ch]。

复杂度分析：

若没有循环周期的概念，每个查询都需要映射完整的 k 次，复杂度为 O(|s| * k)。引入循环周期后，可以将复杂度优化为 O(|s| * min(k, zhouqi[ch]))，其中 min(k, zhouqi[ch]) 的值通常很小。最终评测结果如下：



4.3 题三

4.3.1 题目描述

题目描述

但是，`patch` 程序在处理 `diff` 的输出时，对格式的要求可以较为宽松。例如，它可以允许块内有注释，也可以允许块的行号与实际原文件的行号不匹配。这是因为，在实际应用中，`diff` 生成后，源文件可能经历了其它的变更，导致行号出现了挪动。`patch` 程序的具体工作过程是：

1. 读取全部输入，将 `#` 开头的行视为注释，并移除；
2. 寻找 `@` 开头的行，并将该行至下一个 `@` 开头的行（或文本结尾）之间的内容视为一个块，如果没有找到 `@` 开头的行，则认为补丁损坏；
3. 从前到后依次对每个块：
4. 解析第一行，检查其格式是否正确，如果不正确，则认为补丁损坏；
5. 解析出 `NN`、`MM`、`nn`、`mm`，其中忽略 `nn`；
6. 如果这个块不是第一个块，检查 `NN` 是否不小于前一个块的 `NN` 与 `MM` 之和，如果不是，则认为补丁损坏；
7. 解析其余行，如果这些行中存在不是以 `-`、`+`、空格开头的行，则认为补丁损坏；
8. 将块中所有以 `-` 开头的行和以空格开头的行提取出来，作为原文件的内容片段；
9. 检查原文件的内容片段的行数是否与 `MM` 一致，如果不一致，则认为补丁损坏；
10. 将块中所有以 `+` 开头的行和以空格开头的行提取出来，作为新文件的内容片段；
11. 检查新文件的内容片段的行数是否与 `mm` 一致，如果不一致，则认为补丁损坏；
12. 如果所有块都通过了检查，则对于每个块：
13. 检查是否存在绝对值小于 `MM` 的整数 δ，使得自原文件的第 `NN` + δ 行开始的 `MM` 行，与块的原文件内容片段完全匹配。如果不是第一个块，还需满足 `NN` + δ 不小于前一个块的 `NN` 与 `MM` 之和，即满足每块对应的原文区域没有重叠。如果不存在这样的 δ，则认为补丁损坏；
14. 如果存在多个这样的 δ，则取绝对值最小的那个，如果仍然存在多个，则取最小的那个；
15. 将原文件的第 `NN` + δ 行开始的 `MM` 行替换为块的新文件内容片段；
16. 将该块和此后的所有块的 `NN` 加上 δ。

补丁应用

时间限制： 1.0 秒

空间限制： 512 MiB

相关文件： 题目目录（样例文件）

刷新

题目背景

西西艾弗岛运营公司的信息技术部门，需要协作开展程序开发和代码审查工作。他们的工作流程是这样的：首先，开发者将代码复制一份，并修改代码副本，从而得到期望的代码。然后，开发者使用 `diff` 工具比较修改前后的代码的区别，并将其输出用邮件发送给代码审查者。审查者收到邮件后，可以直接观察开发者作出的代码变更，并提出意见。反复进行后，当代码审查者对变更满意时，会使用 `patch` 程序，将开发者提出的修改应用到原代码上，从而得到最终的代码。

现在，他们已经可以实现 `diff` 程序，但是需要你帮助他们实现这个 `patch` 程序。

`diff` 的输出称作**补丁**。补丁由一个或多个块组成。每个块包含若干行文本，表示对文件的一处修改。其中第一行以 `@@` 开头和结尾，形如：

```
@@ -NN,MM +nn,mm @@
```

其中 `NN`、`MM`、`nn`、`mm` 表示一个 1 至 9 之间的字符和零个或多个 0 至 9 之间的字符组成的字符串，表示一个正整数。每块的第一行表示原文件和新文件的行号范围。其中，`NN` 表示这处修改在原文件的第 `NN` 行开始（原文件的行号从 1 开始编号）；`MM` 表示这处修改涉及原文件的 `MM` 行；`nn` 表示这处修改，在修改后从新文件的第 `nn` 行开始；`mm` 表示这处修改在修改后，在新文件中有 `mm` 行。

随后会有若干行文本，表示修改的内容。如果一行文本以 `-` 开头，表示这行文本在原文件中被删除；如果一行文本以 `+` 开头，表示这行文本在新文件中被添加；如果一行文本以空格开头，表示这行文本在原文件和新文件中都存在，未发生变化。因此，一个块中所有以 `-` 开头的行和以空格开头的行的总数，应该等于 `MM`；一个块中所有以 `+` 开头的行和以空格开头的行的总数，应该等于 `mm`。一处修改的描述中，可以适当包含若干不变的行，以便确定修改的上下文。

4.3.2 算法分析

数据结构以及变量解释：

ss[N]：存储输入的初始文本文件，每行为一个字符串。

v：存储需要插入的新内容。

yuan：存储模板匹配文本。

a[10]：规则参数，通常包括起始行、匹配长度等。

flag：用于标记解析规则的状态，1 表示进入规则解析阶段。

s：用于存储当前读取的行。

```
int siz=yuan.size();

rep(i,1,n)
{
    bool flag=1;
    int cnt=0;
    rep(j,i,i+siz-1)
    {
        if(yuan[cnt  ]!=ss[j])
        {
            flag=0;
            break;
        }
        cnt++;
    }
    if(flag)
    {
        a[1]=i;
        break;
    }
}

 bool fla=0;
 rep(i,1,n)
 {
     if(i<a[1]  || a[1]+a[2]-1<i)
     {
         cout<<ss[i]<<endl;
     }
     else if(fla==0)
     {
         fla=1;
         for(auto j:v)
         {
             cout<<j<<endl;
         }
     }
  }

 return 0;
}
```

1. 查找模板匹配位置：

编历文本文件的每一行，从某一行开始逐行匹配 yuan 中的内容，找到完全匹配的起始位置并存储到 a[1]。

2. 文本替换：

遍历整个文本，非匹配部分直接输出。匹配部分仅输出替换内容（替换内容从 v 中读取）

复杂度分析：

输入阶段

读取 n 行初始文本，时间复杂度为 O(n)。

规则解析，时间复杂度为 O(m)（m 为规则行数）。

模板匹配

最坏情况下，需遍历 n * siz 次，复杂度为 O(n * siz)。

输出阶段

遍历 n 行文本，时间复杂度为 O(n)。

但是算法设计的并不够严密完整，题目要求并没有完全达到，如下为测评结果

125822	王志豪	补丁应用	g++	Wrong Answer	30	2024-11-30 16:06:13

4.4 题四

4.4.1 题目描述

通讯延迟

时间限制： 1.5 秒

空间限制： 512 MiB

相关文件： 题目目录（样例文件）

刷新

题目描述

给定二维平面上 n 个节点，以及 m 个通讯基站。第 i 个基站可以覆盖以坐标 (x_i, y_i) 为中心、$2r_i$ 为边长的正方形区域，并使正方形区域内（包含边界）所有节点以 t_i 单位时间的延迟进行相互通讯。

求节点 1 到 n 的最短通讯延迟。

输入格式

从标准输入读入数据。

第一行包含空格分隔的两个正整数 n、m；

接下来 n 行，每行两个整数 x_i, y_i，代表第 i 个节点的坐标；

接下来 m 行，每行四个整数 x_j, y_j, r_j, t_j，代表第 j 个通讯基站的坐标，通讯半径与通讯延迟。

输出格式

输出到标准输出。

输出一行，即节点 1 到 n 的最短通讯延迟；如果无法通讯，则输出 Nan。

4.4.2 算法分析

该程序通过给定的 n 个点和 m 个区域，构建一张有向带权图。使用 **Dijkstra 算法** 求解从起点（节点 1）到终点（节点 n）的最短路径。如果到达终点的路径不存在，输出 Nan，否则输出最短路径的权重和。

数据结构与变量解释：

```
struct node
{
    int id;
    ll n_dis;
    node(int b,ll c)
    {
        id=b;
        n_dis=c;
    }
    bool operator < (const node &a) const
    {
        return n_dis>a.n_dis;
    }
};

struct edge
{
    int to;
    ll w;
    edge(int a,ll c)
    {
        to=a;
        w=c;
    }
};
```

1. 节点 node：

表示一个节点及其到起点的距离 n_dis。并且重载了小于运算符 <，使其适用于最小堆（priority_queue）。

2. 边 edge：

表示一条从一个节点到另一个节点的边。包括目标节点 to 和权重 w。

3. 邻接表存储图：

使用 vector<edge> e[N] 存储图，其中 e[i] 表示与节点 i 相连的所

有边。

```
rep(i,1,m)
{
    ll xx,yy,r,t;
    cin>>xx>>yy>>r>>t;
    vector<int > aa,bb;
    rep(j,1,n)
    {
        if(    xx-r<=x[j]&&x[j]<=xx+r&&yy-r<=y[j]&&y[j]<=yy+r)
        {
            aa.pb(j);
            bb.pb(j);
        }
    }
    for(auto ii:aa)
    {
        for(auto jj:bb)
        {
            if(ii==jj)
            {
                continue;
            }
            e[ii].pb(edge(jj,t));
        }
    }
}
```

1. 图的构建：

对于每个区域 (xx, yy, r, t)，判断哪些点落入该区域内。区域内的所有点相互连接，权重为 t。

```
void di()
{
    int s=1;
    rep(i,1,n)
    {
        dis[i]=inf;
        done[i]=0;
    }
    dis[s]=0;
    priority_queue<node > q;
    q.push(node(1,0));
    while(q.size())
    {
        node u=q.top();
        q.pop();
        if(done[u.id])
        {
            continue;
        }
        done[u.id]=1;
        for(int i=0;i<e[u.id].size();i++)
        {
            edge y=e[u.id][i];
            if(done[y.to])
            {
                continue;
            }
            if(dis[y.to]>y.w+u.n_dis)
            {
                dis[y.to]=y.w+u.n_dis;
                q.push(node(y.to,dis[y.to]));
            }
        }
    }
}
```

2. Dijkstra 算法：

初始化所有节点到起点的距离为 inf，起点距离设为 0。使用优先队列（最小堆）维护当前未访问节点中的最短路径。每次从堆中取出距离最小的节点，更新其邻接节点的最短路径。如果最终到达终点的距离仍为 inf，表示无法到达。

复杂度分析

1. **图的构建：**

每个区域最多遍历 n 个点，判断是否在区域内的时间复杂度为 O(m * n)。

区域内点的两两连边操作复杂度为 O(n^2)（最坏情况下）。

2. **Dijkstra 算法：**

使用优先队列实现，时间复杂度为 O(E log V)，其中 E 是边数，V 是点数。

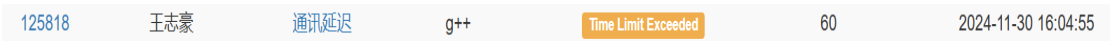
总体复杂度：

图的构建：O(m * n + n^2)

Dijkstra：O(E log V)

在最坏情况下，时间复杂度为 **O(m * n + n^2 + E log V)**。

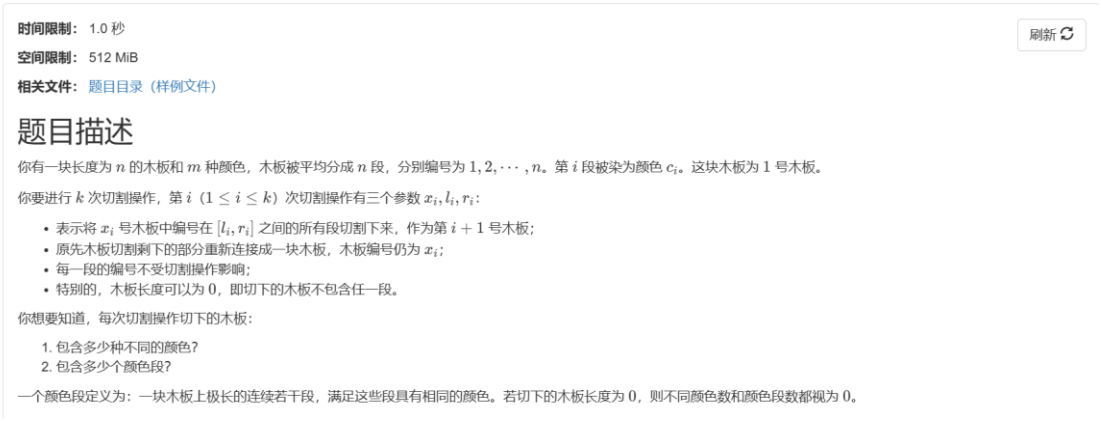
测评结果如下,结果依旧时间超限:



4.5 题五

4.5.1 题目描述

木板切割



时间限制： 1.0 秒

空间限制： 512 MiB

相关文件： 题目目录（样例文件）

刷新

题目描述

你有一块长度为 n 的木板和 m 种颜色，木板被平均分成 n 段，分别编号为 $1, 2, \cdots, n$。第 i 段被染为颜色 c_i。这块木板为 1 号木板。

你要进行 k 次切割操作，第 i $(1 \leq i \leq k)$ 次切割操作有三个参数 x_i, l_i, r_i：

- 表示将 x_i 号木板中编号在 $[l_i, r_i]$ 之间的所有段切割下来，作为第 $i+1$ 号木板；
- 原先木板切割剩下的部分重新连接成一块木板，木板编号仍为 x_i；
- 每一段的编号不受切割操作影响；
- 特别的，木板长度可以为 0，即切下的木板不包含任一段。

你想要知道，每次切割操作切下的木板：

1. 包含多少种不同的颜色？
2. 包含多少个颜色段？

一个颜色段定义为：一块木板上极长的连续若干段，满足这些段具有相同的颜色。若切下的木板长度为 0，则不同颜色数和颜色段数都视为 0。

4.5.2 算法分析

数据结构与变量解释：

vector<node> v[N]：

1. 维护每个操作中有效的节点集合。

2. 每个节点包括 id（位置）和 c（颜色）。

vector<pii> p[N]：

1.用于存储连续区间 [left, right]。

2.优化场景下，直接使用连续区间进行分割和合并操作，避免逐个位置判断。

col[N]：

记录每个位置的颜色值。

sum[N]：

维护颜色段数的前缀和。

set<int>：

用于存储不同颜色值，以便快速统计颜色种类。

```
rep(i,1,n)
{
    int cc;
    cin>>cc;
    col[i]=cc;
    if(cc>i)
    {
        flagb=0;
    }

    sum[i]=sum[i-1];
    if( col[i-1]!=col[i])
    {
        sum[i]++;
    }
```

1. 初始化

读取颜色数组 col。

如果颜色值不满足条件（如 col[i] > i），将标志 flagb 设置为 0。

计算颜色段的前缀和 sum：

如果当前位置颜色与前一个位置颜色不同，段数加 1。

```
if (flagb) {
    p[1].pb({1, n});  // 初始区间

    rep(i,1,k) {
        vector<pii> temp;
        for (auto j : p[x[i]]) {
            int leftt = j.first, rightt = j.second;
            if (rightt < l[i] || r[i] < leftt) {
                temp.pb(j);  // 区间完全不相交
            }
            else {  // 区间相交，分割
                if (l[i] <= leftt) {
                    if (r[i] < rightt) {
                        p[i+1].pb({leftt, r[i]});
                        temp.pb({r[i]+1, rightt});
                    } else {
                        p[i+1].pb({leftt, rightt});
                    }
                }
                else {
                    if (r[i] < rightt) {
                        p[i+1].pb({l[i], r[i]});
                        temp.pb({leftt, l[i]-1});
                        temp.pb({r[i]+1, rightt});
                    } else {
                        p[i+1].pb({l[i], rightt});
                        temp.pb({leftt, l[i]-1});
                    }
                }
            }
        }
        p[x[i]] = temp;

        // 统计颜色段数
        int duan = 0;
        int lasttc = 0;
        for (auto j : p[i+1]) {
            duan += col[j.second] - col[j.first] + 1;
            if (lasttc == col[j.first]) {
                duan--;
            }
            lasttc = col[j.second];
        }
        cout << duan << " " << duan << endl;
    }
    exit(0);
}
```

2. 优化方案 (FlagB)

如果 flagb = 1，启用区间合并优化：

通过区间 [l[i], r[i]] 的关系对连续段进行分割和合并。

分为以下几种情况：

区间完全不相交：直接保留原区间。

区间部分相交或完全包含：分割当前区间，并记录操作影响的区间。

计算段数和颜色种类：

遍历当前的区间集合 p[i+1]，统计颜色段数和总颜色种类。

```
rep(i,1,k) {
    vector<node> temp;
    for (auto j : v[x[i]]) {
        if (l[i] <= j.id && j.id <= r[i]) {
            v[i+1].pb(j);  // 符合条件的节点
        }
        else {
            temp.pb(j);
        }
    }
    v[x[i]] = temp;

    set<int> yanse;  // 记录颜色种类
    int duan = 0, lastt = 0;

    for (auto j : v[i+1]) {
        if (j.c != lastt) {
            duan++;
            lastt = j.c;
        }
        yanse.insert(j.c);
    }
    cout << yanse.size() << " " << duan << endl;
}
```

3. 普通方案

如果无法优化（flagb = 0），逐个节点检查：

遍历 v[x[i]]，保留在 [l[i], r[i]] 范围内的节点。

用 set 记录不同颜色，用 lastt 判断颜色段数。

时间复杂度分析

1. 优化方案：

   每个操作对区间进行分割和合并，最坏情况下复杂度为 O(k * log n)。

   遍历区间集合时复杂度为 O(n)。

2. 普通方案：

   每次操作遍历当前节点集合 v[x[i]]，最坏情况下复杂度为 O(k * n)。

测评结果如下,时间超限：

125812	王志豪	木板切割	g++	Time Limit Exceeded	65	2024-11-30 16:01:22