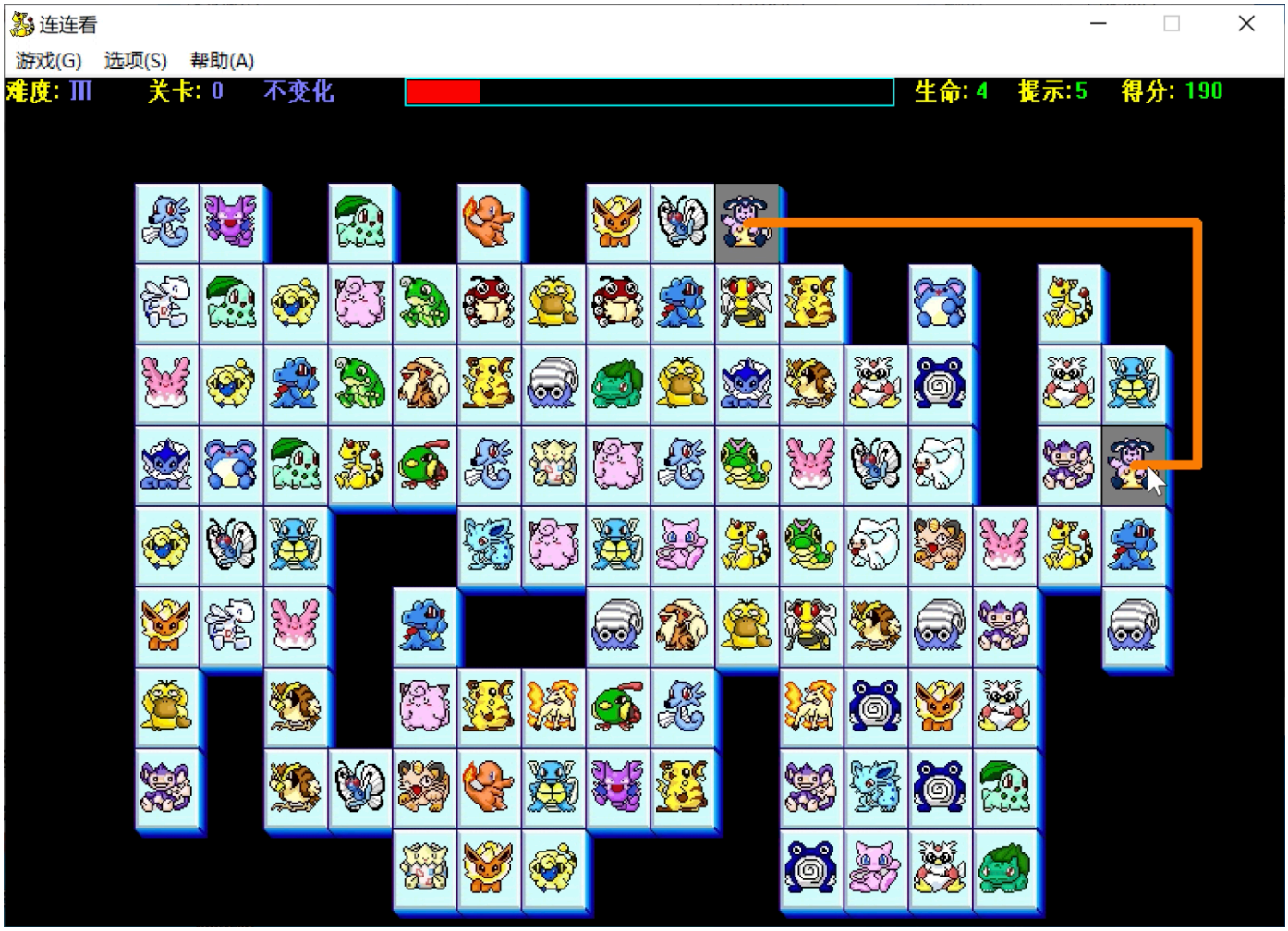
Project 1: QLink

连连看



（大家不用按照上面这个图来做，这只是曾经这个游戏的截图，我们不需要模仿它）

QT GUI 基础，从 AnalogClock 到推箱子

GUI 编程的大体思路

大多数 GUI 编程，是基于事件的（Event-driven）。在这种事件驱动的编程模型中，存在事件和处理函数两个概念：

- 事件（Event），如鼠标移动、鼠标点击、键盘输入、窗口大小改变等，用户也可以自己定义事件。
- 事件处理函数（Event Handler），程序员可以将函数与事件进行绑定，当某个事件触发之后，系统会自动调用对应的处理函数。

简单来说，GUI 程序的大体思路是这样的：

```
int main()
{
    // 准备一些基本的界面
    prepare_display();
    // 为事件绑定处理函数，这部分往往和准备界面是混在一起的
    // 这里单独提出来，主要是为了突出事件和处理函数的重要性
```

```
    bind_handlers_to_events();

    // 开始进入主循环，只要程序不退出，该循环不会退出
    for (;;) {
        // 等待一个事件发生
        event = wait_for_event();
        // 获取绑定在该事件上的处理函数
        handlers = get_handlers(event);
        // 调用这些绑定在事件上的处理函数
        for (handler in handlers) {
            handler();
        }
    }
    return 0;
}
```

因此，我们要实现一个 GUI 程序，重要的是准备界面和写处理函数。

QT 中的事件和处理函数

在 QT 中，有 Signal、Slot 和 `connect` 的概念，其中 Signal 和 Slot 对应着前面的 Event 和 Event Handler，`connect` 则将两者进行绑定。

```
// 一个 Label 实例，类似于文本框，但是只用来显示文字，用户不能修改其中的内容
QLabel *label = new QLabel;
// 一个滚动条实例
QScrollBar *scrollBar = new QScrollBar;
// 下面的这个 connect 增加了这条绑定关系：
// 当这个滚动条上发生了 valueChanged(int) 这个事件的时候（也就是滚动条被拖动的时候），
// label 对象上的 setNum(int) 函数被调用
// 最后的效果，是用户拖动滚动条，label 里面会动态显示当前滚动条的位置（一个数字）
QObject::connect(scrollBar, SIGNAL(valueChanged(int)),
                 label,  SLOT(setNum(int)));
```

Practice 1

请阅读给出的代码和注释，将其运行起来

Practice 2：增加秒针

1. 将分针的颜色改为蓝色
2. 增加秒针，颜色为红色，长度大于分针长度，形状为任意四边形

- 请注意，你的程序是写给用户的，因此不要只为了完成作业而脱离实际，请考虑让用户能看懂这个是秒针

Practice 3：时间暂停

1. 增加按键功能，按下 P 键则钟表的时间停止

Qt 中的按键也是一个事件，由于非常常用，Qt 已经帮我们做好了处理函数：

```
[virtual protected] void QWidget::keyPressEvent(QKeyEvent *event)
```

我们的 AnalogClock 类是 QWidget 的子类，因此可以 override 该函数，来处理键盘按键事件。具体怎么用，怎么获取按下的是哪个键，还需要你去看一下文档呦。

2. 再按 P 键恢复

- 此处的实现不需要考虑时间连续性，因此在恢复时表针会发生跳跃，显示真正的当前时间
- 思考：如何模拟时间暂停，按 P 键恢复之后，时间从暂停时的下一秒开始?

Practice 4：移动的人物角色

1. 在屏幕上，画一个圆圈表示人物角色
2. 按键 WASD 控制该圆圈上下左右移动
3. 当人物在边界走时，可以从另外另外一端的边界出来

- 思考：如何让用户自动向前走，WASD 只是控制自动移动方向?

Practice 5：推箱子

1. 在 4 基础上，再画一个方块表示箱子
2. 当人物移动到箱子旁边，并向箱子方向再次移动时，会推动箱子一起移动

通过上面的练习，你应该已经掌握基本的 QtGUI 编程的能力。 一些没有涉及到的操作和用法，可以在此后的实践中进行学习，请善用 QT 的帮助文档和搜索引擎。

QLink

在连连看游戏中，会有一个地图，地图上有许多不同种类的方块，通过将相同种类的两个方块相连，可以将这两个方块消除，用户获得分数。

在整个连连看的过程中，除了处理用户的操作之外，还有几个比较特殊的部分特别需要注意：

- 随机地图的生成
- 判断两个方块是否可以通过两次以内的折线进行连接
- 判断剩余方块是否还有解

除此之外，具体的功能要求如下：

RPG 机制

不同于传统的连连看，我们的 QLink 使用 **RPG 模式**进行，即玩家需要控制一个角色在地图的空地上移动（角色显示可自行选择）。

- 激活：当角色处于方块旁且再次向方块方向移动，会激活该方块（请使用某种方式表示该方块被激活）。
- 消除：如果此次激活的方块和上次激活的方块是同种类，且可以通过两次以内的折线连接，则该两个方块被消除，玩家获得分数。（请绘制出将两个方块连接在一起的折线）否则，上次激活的方块被自动变为未激活状态，换句话说，每个角色在地图中只有 0 个（游戏刚开始时，或刚刚消除完一对方块时）或者 1 个激活的方块。

计分

不同种类的方块可以有不同的分值，具体规则可以自行制定。界面中应时刻显示玩家的分数。

倒计时和游戏结束

有两个情况可以导致游戏结束：

1. 倒计时结束；
2. 没有可消除的方块对（所有方块均被消除也属于这一种）。

界面中应时刻显示游戏的倒计时。

开始菜单

至少包括以下按钮：

- 开始新游戏
    - 可选择游戏模式：单人模式、双人模式（具体看后文）
- 载入游戏
- 退出游戏

单人模式

游戏开始时，会随机生成地图，并随机玩家角色位置。随后玩家可控制角色移动，以激活和消除方块。

道具

道具通过随机方式出现在地图的空地上，当角色与道具出现在同一位置时，该角色触发道具效果，道具消失。

- +1s：延长剩余时间 30s
- Shuffle：所有方块位置重排
- Hint：10s 内会高亮一对可能链接的方块，被消除后会高亮下一对，直到 10s 时间结束
- Flash：5s 内允许通过鼠标单击移动角色位置，角色移动到的位置必须通过空地可到达，否则点击不产生任何效果。如果点击到方块，且角色可以移动到该方块旁，则角色移动到该方块旁，且该方块被激活。如果方块四周有多个位置可以让角色停留，则角色移动到其中任何一个位置均可。

双人模式

两个玩家的两个角色在相同的地图上进行游戏，以结束游戏时双方的分数决定谁为赢家。

道具

在单人模式的基础上，增加：

- Freeze：对手 3s 内无法移动
- Dizzy：对手 10s 内移动方向颠倒（上下左右颠倒）

此外，

- Hint 道具的效果对两个玩家均可见；
- +1s 道具的效果对两个玩家均有效。
- 双人模式下，没有 Flash 道具

暂停和存档

- 在暂停时，可以保存游戏（Save）和载入游戏（Load）
- 保存游戏会将当前游戏的所有状态以任意格式保存到磁盘上的文件
- 载入游戏时，读取文件，并从中恢复状态

分阶段任务

Step 1: 地图生成和展示

- 生成一个M*N的地图
  - M和N可以自定义
- 生成不同类型的箱子，并在界面上显示出类型的不同
  - 实现至少3种类型的箱子

Step 2: 小人能在地图上移动，并能与边缘的block交互

- 小人能在地图上自由移动
- 小人能够对箱子进行操作，操作成功后界面上要有反馈
  - 小人对箱子的操作包括：靠近，选定

Step 3: 选中方块后的判定逻辑

- 将判定逻辑单独作为一个函数编写
  - **单元测试要求**：对判定逻辑的函数，需要额外编写的单元测试，对正常运行情况和边界情况都进行测试
  - 单元测试的样例在文档末尾给出
- 每次小人选中2个方块时，运行判定函数，输出判定结果，显示到界面上
  - 判定结果包括：不可消除，消除
  - 若消除，则将对应箱子从地图中删去

- 每次方块消除后，都要判断当前地图是否可解，若无解则显示在界面上，并使得游戏结束

Step 4: 时间和分数

- 在地图上显示倒计时，并实时更新
- 若倒计时完成，则显示游戏结束

Step 5: 道具

- 先实现+1s道具
  - 随机在地图上生成道具
  - 判断人物是否接触到道具
- 实现其他道具
  - **Shuffle 必须实现，Hint 和 Flash 至少选择一个实现**
  - Hint：遍历整个地图上现存的方块，找到一对能消除的方块高亮，消除后找到并高亮下一对，直到持续时间结束
  - Flash：鼠标单击触发角色位置变化，角色位置定位到鼠标点击的位置

Step 6: 暂停和存档

- 暂停
  - 暂停游戏时间，不允许玩家操作
- 存档
  - 将游戏状态输出至文件，如人物位置、剩余时间、剩余方块等。并且可以读取这个文件，恢复游戏状态

Step 7: 实现双人模式

- 实现双人模式
  - 在游戏中添加第二个角色，使用不同按键操作，其他逻辑相同
  - 注：将角色实现成一个类会让代码更简洁。两个角色就对应类的两个实例
- **双人模式下的道具为可选项（不计如最终分数）**

Step 8: 开始菜单

- 制作一个开始菜单
  - 可以是任意形式
    - 几个按钮
    - 一个列表包含多个选项
    - 一个单独的窗口
    - 和游戏属于同一个窗口的不同状态
    - ……

评分标准

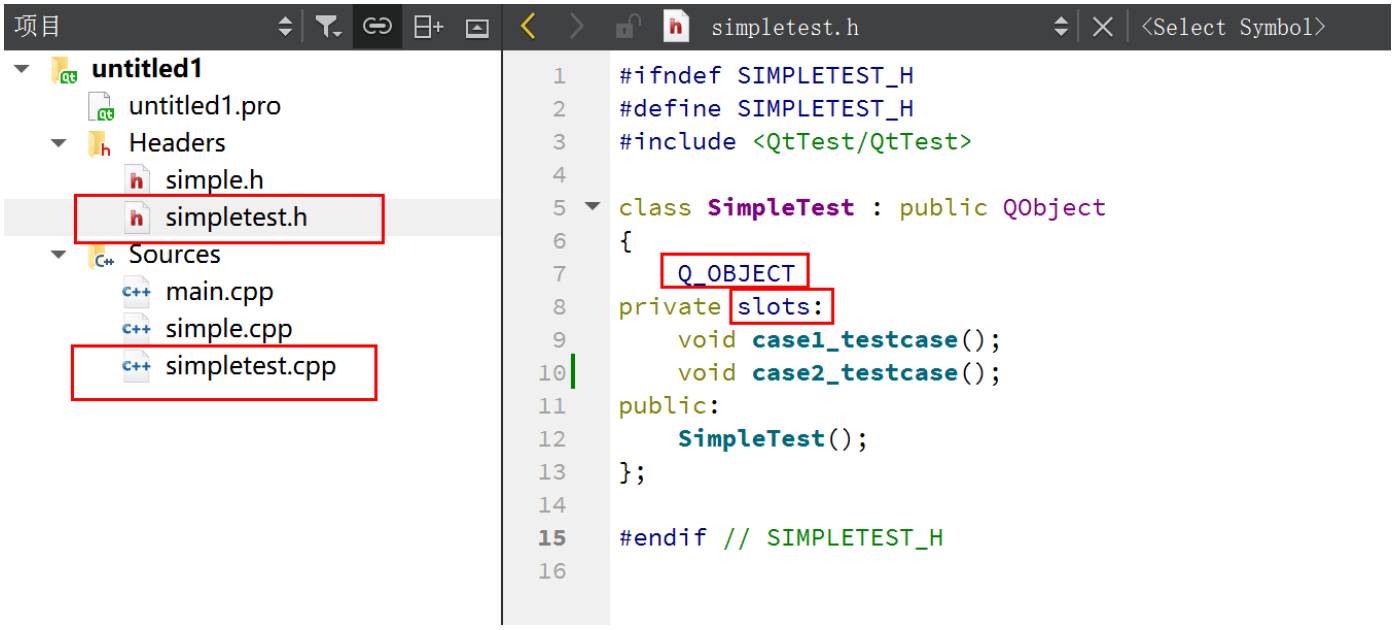
- 以上 8 个步骤，每个 10 分
- 此外，单元测试占 10 分，代码质量（风格和注释等）占 10 分

附录1：测试用例样例

Step1

创建测试类头文件.h

新建的测试类`SimpleTest`，必须要继承`QObject`，才能在测试的时候调用QT提供的测试框架

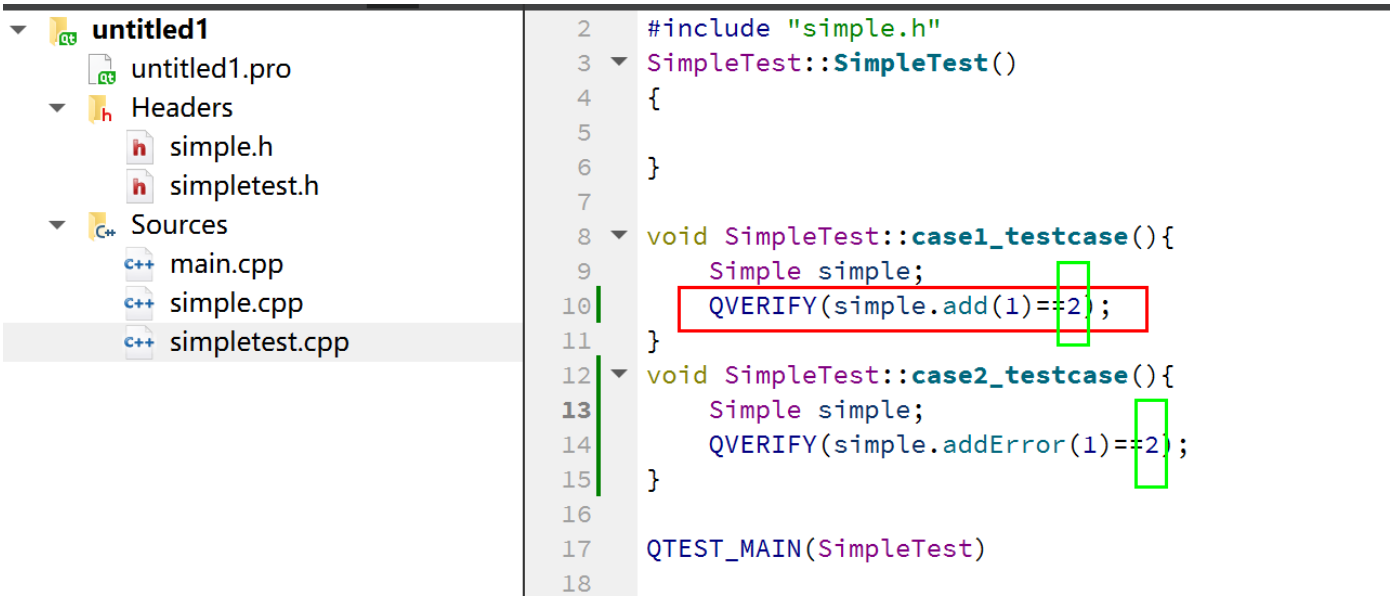


Step2

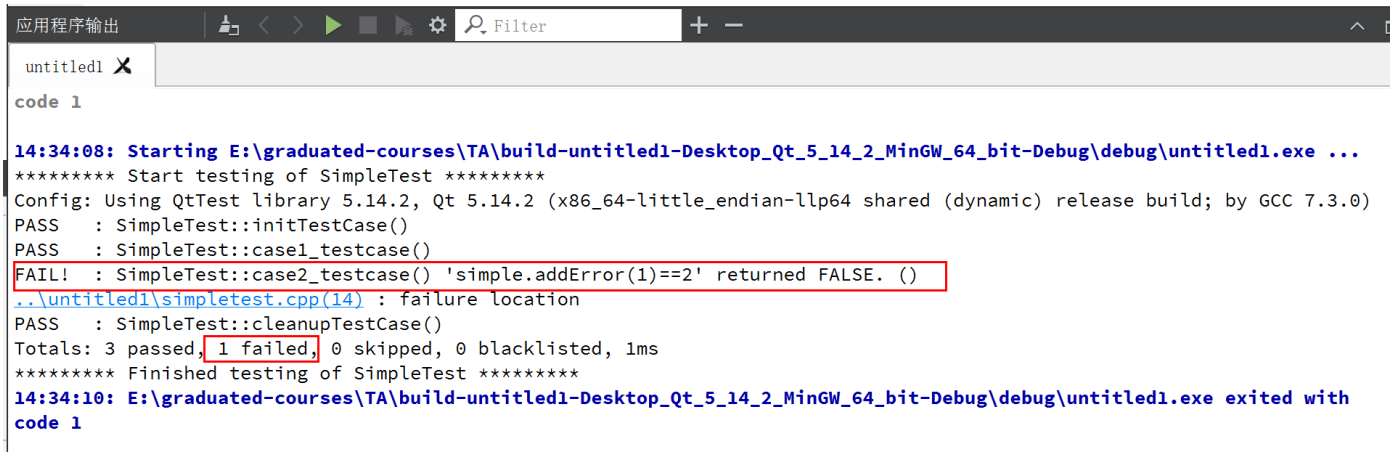
创建simpletest.h的实现.cpp文件

通过调用`QVERIFY`函数来测试自己的目标函数，其中用绿色的框圈出来的是预期的正确结果（一般可以通过手算来获取，一定要保证是正确的），等于号左侧是测试目标函数。

`QTest_Main`是QT自带测试框架中所要求的入口函数，功能相当于`main()`



上图来说，add是一个实现正确的，目标为加一的函数；addError是一个实现错误的，目标为加一的函数，运行的结果如下所示：



PS: 如果项目启动出错的话，比较一下自己的.pro文件和助教的.pro文件配置，看看有没有缺少的内容

什么是好的测试

一个测试应该覆盖正常的执行流程以及corner case（也就是异常处理）。以除法为例，假设我们需要一个除法 `divide(a,b)`

那么测试用例应该包括两个部分：

1. 正常的执行流程

   e.g. `QVERIFY(divide(2,1)==2)`

2. 异常处理

   e.g. `QVERIFY(divide(2,0)==NAN)`

一个优秀的测试尤其需要注意异常情况的处理。

此外正常执行流程中的每一个逻辑分支都应该尽量被覆盖到，比如

- 若有 `if-else` 的情况，应该两种执行路径都被测试到。

如果想要更详细的测试，可以自行查询软件测试分类，尝试去满足以下类别下的测试：

1. 语句覆盖【这一点一定需要做到，其余随意】 每个语句至少被执行一次。
2. 判定覆盖（分支覆盖） 每个判定语句都至少取真一次、取假一次。
3. 条件覆盖 每个条件至少有一次为真。
4. 条件判定覆盖 每个判定语句都至少取真一次、取假一次。且每个条件至少有一次为真。
5. 条件组合覆盖 条件的各种组合至少出现一次。 （全为真、全为假、一部分为真）
6. 基本路径覆盖 所有可能的路径至少执行一次。 （真真、假假、真假、假真）

附录2：QLink 代码规范

下面是本项目中会被检查的代码规范，下面未列出的不做要求。

命名规范

所有命名使用英文单词（不要使用拼音）或其缩写，以及一些常用的简写变量。

举例：

```
int counter = 0;
int score;

int i, j, k;
int tmp;
int cnt;
int sum;
```

类

- 类名：每个单词的首字母大写
- 变量和函数名：使用驼峰格式

```
class Point {
    int x;
    int y;
};

class TableElement {
    int numRows;
    int numCols; // Number of columns
public:
    void setAtCell(int row, int col);
    void getAtCell(int row, int col);
};
```

- 除非特殊情况，否则头文件开头使用

```
#pragma once
```

思考：这个是做什么用的?

示例代码

```
//
// getScore 接受课程号(classId)作为参数，返回对应课程的分数
//  未找到时返回-1
//
int Student::getScore(int classId, const vector<Class> &allClasses) const
{
    if (classId < 0 || classId >= numClasses) {
        // 无效的课程 ID
        return 0;
    } else {
        vector<Class>::Iterator cl;
        // 遍历所有课程，找到特定class id的课程分数
      for (cl = allClasses.begin(); cl != allClasses.end(); ++cl) {
            if (cl->id == classId) {
              return cl->getScore(this);
            }
        }
        // 未找回该 class
        return -1;
    }
}
```

一些具体的注意点

- 大括号位置：
  - if、else、for、while 等 在同一行末尾
  - 函数定义末尾的大括号在下一行
- 缩进：每一级 4个空格缩进
- 每个两元运算符（即 `X op Y` 形式中的 `op`）左右各有空格，比如 `+`, `>=`, `=` 等
- 所有不会被修改的参数需要按照const引用的方式传递 (int, double等基本类型可以直接传递，不需要使用const引用)
  - 例如下方示例中，DIYClass的用法

```
int testFunc(int param1, const DIYClass &class)
{
    ...
}
```

函数头格式

- 函数注释要能表示函数具体行为，要能涵盖每个参数的含义和作用，解释返回值的含义

函数长度要求

原则上每个函数不超过 50 行。