

LemonLime 用户手册

浮尘 ii*, iotang, Coelacanthus

2021 年 2 月 17 日



目录	2
----	---

目录

1 历史	4
2 Lemon 与开源意志	6
3 版本兼容性	7
4 安装	8
4.1 支持的系统	8
4.2 安装方法	8
4.2.1 Windows 和 macOS	8
4.2.2 Linux	8
5 设置	10
5.1 常规设置	10
5.2 编译器设置	11
5.3 视觉设置	16
6 比赛	18
6.1 添加新试题	19
6.2 自定义校验器 (SPJ) 说明	21
6.2.1 校验器实例	21
6.2.2 使用 testlib	22
6.3 关于非传统型试题	23
6.3.1 提交答案题	24
6.3.2 交互题	24
6.3.3 通信题	24
6.4 添加新测试点	24
6.5 批量添加测试点	25
6.6 增强测试点调整器	26
6.7 自动添加试题	28
7 测试	29
7.1 测试情况对话框	29
7.2 颜色	30
7.3 整理文件	30
7.4 跳过这一题	30

目录	3
8 导出成绩	31
9 比赛统计	32
9.1 总览	32
9.2 题目	32
9.3 导出统计信息	33
10 常见问题及回答	34
10.1 测评时使用更多栈空间	34

1 历史

[LemonLime](#) 项目是 [LemonPlus](#) 的二次开发。

2011 年，Jia Zhipeng 开发完成了 Lemon，一个开源的评测工具。

虽然我们并不知道他确切的动机，

——是想弥补 Linux 下没有评测工具的缺陷，还是想造出一个 Cena 的对手，

不管怎样，他的愿望达成了，Lemon 的历史也从此开始。

现在（2020 年）他已经是一名得克萨斯大学奥斯汀分校的 2 年级博士生了，祝他的学业生涯顺利。

2018 年，在浮尘 ii* 的努力下，Lemon 进化为 LemonPlus。

LemonPlus 成功兼容了 Qt 5，

并且新增了适合新时代 OI 的功能，移除了一些不稳定的、过时的功能。

奈何 OI 残酷，岁月更迁，物是人非，浮尘 ii* 在 2019 年退役了，

LemonPlus 的开发也相应地结束了。

2019 年，iotang 开始魔改 Lemon。

起初，是想给 Lemon 增添一点用户体验——

添加更多的颜色、优化界面，以及修改令人崩溃的重测逻辑，

但是误打误撞地在 Github 上发现了 LemonPlus。

可怜的 iotang 当时还不会用 Pull Request，所以 LemonLime 诞生了。

LemonLime 继承了 LemonPlus 和 LemonMt（这个大概只有长郡人才会知道）的优点。

放眼望去，Lemon 的历史就像是传火。

Lemon 开发要想继续，就必须燃烧 Oier 的灵魂。

2020 年，随着政策巨变，iotang 也在退役的路上了。

虽然退役并不代表开发停止，但是开发 Lemon 的热情将再也不能体会到了。

也只能可惜 iotang 不够强了吧。

2020 年，因为功能欠缺的 qmake，

Ceolacanthus 开始了把 LemonLime 从 QMake 迁移到 CMake 的路程。

又因为不想每次都编译，她开始改造 LemonLime 的 CI，

试图完成自动编译和打包。

2020 年底，她成功的让 LemonLime 兼容了 Qt6。

当然，这一切都还没完成。

本篇用户手册是仿照 LemonPlus 的用户手册进行编写的。

2 Lemon 与开源意志

LemonLime 使用 [GPLv3](#) 协议。也就是说，LemonLime 是自由软件。

自由软件是什么？为什么 LemonLime 选择成为自由软件？

自由软件意味着使用者有运行、复制、发布、研究、修改和改进该软件的自由。

自由软件是权利问题，不是价格问题。

要理解这个概念，你应该考虑“free”是“言论自由（free speech）”中的“自由”，而不是“免费啤酒（free beer）”中的“免费”。

更精确地说，自由软件赋予软件使用者四项基本自由：

- 不论目的为何，有运行该软件的自由（自由之零）。
- 有研究该软件如何工作以及按需改写该软件的自由（自由之一）。
- 有重新发布拷贝的自由，这样你可以借此来敦亲睦邻（自由之二）。
- 有向公众发布改进版软件的自由（自由之三），这样整个社群都可因此受惠。

不管是 Lemon，还是它的后继者 LemonPlus 和 LemonLime，都恪守着自由软件的意志。

这是 LemonPlus 和 LemonLime 诞生的保障，也是 Lemon 不断延续自己的生命的保障。

Lemon，以及它的后继者们，拥抱开源。

我们希望，Lemon 可以为全世界的 OIer 带来福祉。

让我们一起，把 Lemon 变得更好。

3 版本兼容性

LemonLime 近乎完全兼容 LemonPlus，出问题的地方在子任务依赖（为了支持 0 分测试点的让步，不过这个问题将会被解决）。不过这个问题在使用对应平台重新测试后就可以自动解决。

LemonLime 完全兼容 Lemon。

LemonLime 主要使用 Qt 编写，目前兼容 Qt5 和 Qt6，目前支持的最低版本为 Qt 5.5。编译时使用 `-DLEMON_QT6=ON` 参数即可使用 Qt6 编译。

LemonLime 可以使用 GCC, Clang 和 MSVC 进行编译，但是如果在 Windows 下使用 GCC 编译则会遇到一些问题

4 安装

LemonLime 在上文中提及的[仓库地址](#)不仅用来存储源代码，也用来实时发布软件最新版本。

4.1 支持的系统

LemonLime 支持 Windows、Linux 以及 macOS。

4.2 安装方法

4.2.1 Windows 和 macOS

在[发行页面](#)下载相应的压缩包。

不过如果你获得了源码，可以通过 QtCreator 来编译 LemonLime。

4.2.2 Linux

这里以 Manjaro 为例，毕竟 Linux 安装时几乎只有命令名字的区别。

更多的系统请到 README.md 查看。

首先你得安装一些依赖。

```
$ sudo pacman -S gcc ninja qt5-base cmake
```

注意 Manjaro 的 gcc 包含了 g++。

然后用某种途径获得源代码。比如使用 git：

```
$ git clone https://github.com/Project-LemonLime/Project_LemonLime.git
```

进入源代码的目录。

```
$ cd < 源代码的目录 >
```

然后依次执行：

```
$ cmake . -GNinja -DCMAKE_BUILD_TYPE=Release
$ cmake --build . --parallel $(nproc)
```

获得可执行文件 lemon。打开 lemon 就可以运行 LemonLime 了。
也可以执行


```
$ cmake --install .
```

将其安装到系统中，默认安装位置位于 `/usr/local`

同时，在装有 `dpkg/rpm` 的 Linux 系统上可为 CMake 附加 `-DBUILD_DEB=ON` 或者 `-DBUILD_RPM=ON` 参数直接生成 Deb/RPM 包。即：

```
$ cmake . -GNinja -DCMAKE_BUILD_TYPE=Release -DBUILD_DEB=ON  
$ cmake --build . --parallel $(nproc)
```

推荐在直接生成包的时候同时附加 `-DEMBED_TRANSLATIONS=OFF` `-DEMBED_DOCS=OFF` 参数关闭翻译文件和手册文件的嵌入。

5 设置

LemonLime 没有设置编译器时（比如第一次运行）会自动弹出添加编译器的向导，具体的说明请看编译器配置一节。

5.1 常规设置

在工具菜单中选择“设置”，就能看到下图所示的对话框：



各个设置意义如下：

默认分值 新建一个测试点时的默认分值，默认为 10，可以设置的最大值为 10000000。

默认时间限制 新建一个测试点默认的时间限制，默认为 1000 ms，可以设置的最大值为 86400000 ms，即 1 天。

默认额外时间限制比率 新建一个测试点默认的额外时间限制比率，默认为 0.1。因 LemonLime 采用 realtime 作为判定 TLE 的标准，需要适当调整此值以免造成误判，参见[issues/84](#)

默认空间限制 新建一个测试点时默认的空间限制，默认为 512 MB，可以设置的最大值为 16777216 MB。

编译时间限制 测试时允许编译器运行的最长时间，默认为 10000 ms，可以设置的最大值为 86400000 ms，即 1 天。

检验器时间限制 对于使用自定义校验器的试题，测试时允许校验器运行的最长时间，默认为 10000 ms，可以设置的最大值为 86400000 ms，即 1 天。

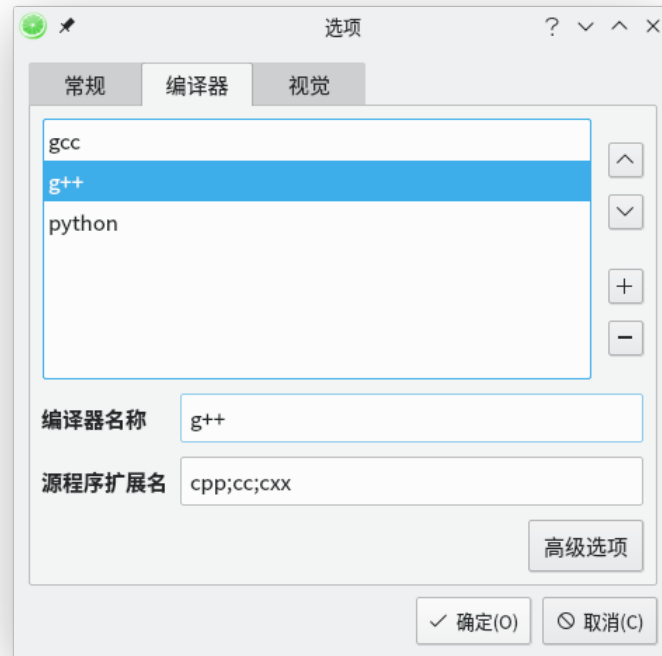
源程序大小限制 测试时可以接受的最大源程序大小，可以设置的最大值为 262144 KB，即 256 MB。(1 KB = 1024 B)

最大重测次数 当测试某测试点时程序运行时间不超过时间限制的 1.1 倍，或超时不超过 100ms 时，LemonLime 会对该测试点进行重测。可以设置的最大值为 12。

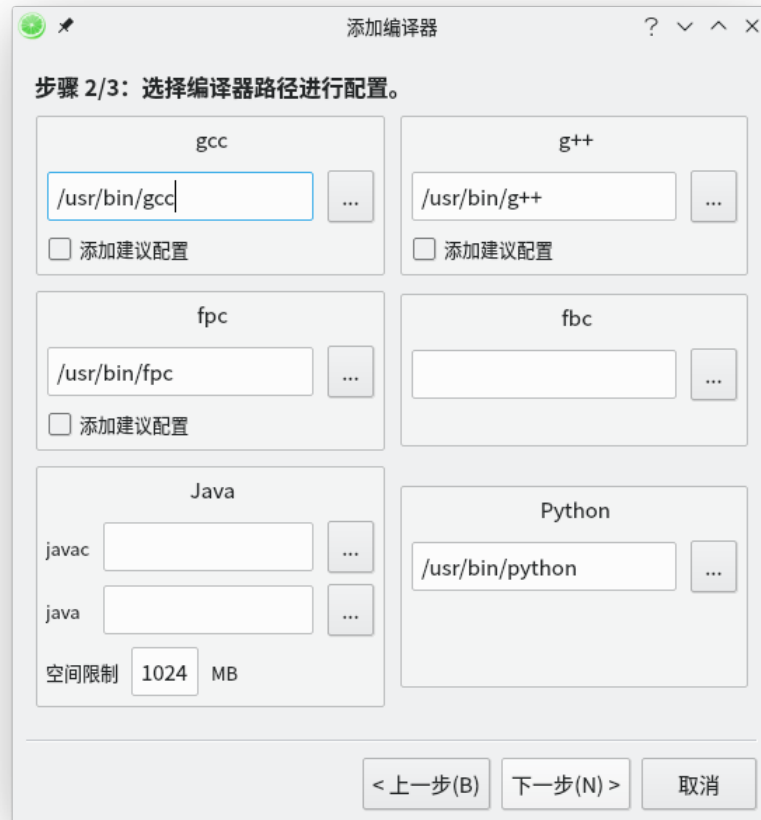
输入、输出文件扩展名 在自动添加试题时扫描的输入和输出文件的扩展名。如果有多个请用 ';' 隔开。每种扩展名中只能包含英文字母和数字，Linux 平台下大小写是敏感的。注意这里的扩展名只供添加测试点时软件搜索文件使用，测试时并不会检查。

5.2 编译器设置

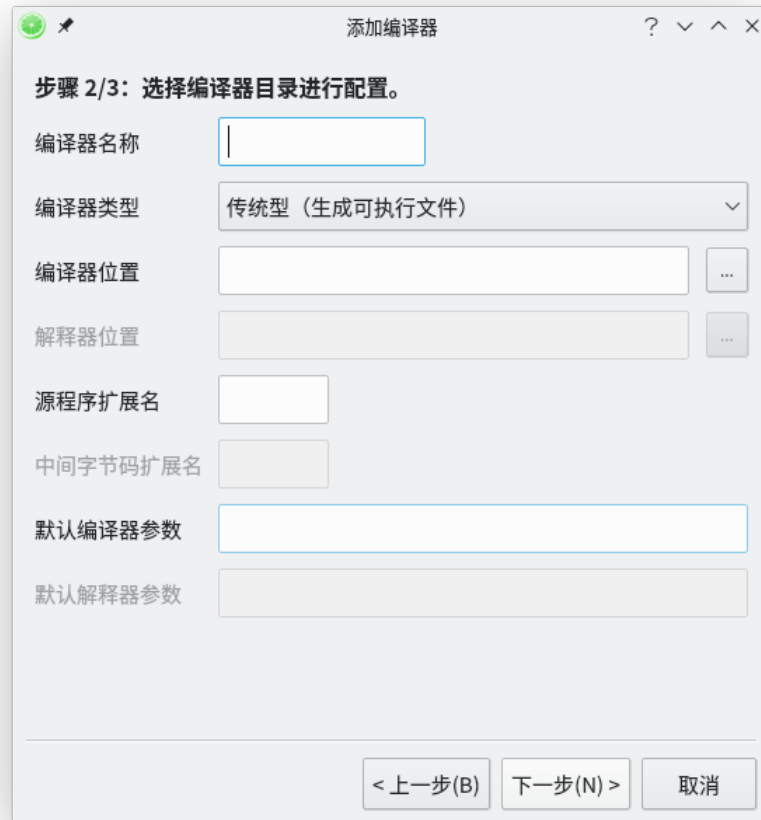
点击上方的“编译器”选项卡就能进入编译器的配置。



按右边的加号就会出现添加编译器的向导，第一步是选择使用预置的编译器配置还是手动配置新的编译器。一般来说，用预置的配置就能够满足大多数需求，第一步只要在需要配置的编译器前打钩，进入下一步后就能看到选择路径的界面。



gcc、g++ 和 fpc 的建议配置中包含了不同标准和不同优化等级的组合。
Java 的内存限制是由 Java 虚拟机限制的，你可以在之后的设置中选择不限内存。
最后一步会让你确认编译器路径是否设置正确。
除了使用内置的六种编译器配置，你也可以选择手动配置新的编译器。



The screenshot shows a dialog box titled "添加编译器" (Add Compiler) with a green icon and standard window controls. The main heading is "步骤 2/3: 选择编译器目录进行配置。" (Step 2/3: Select compiler directory for configuration). The form contains the following fields and controls:

- 编译器名称** (Compiler Name): A text input field.
- 编译器类型** (Compiler Type): A dropdown menu currently showing "传统型 (生成可执行文件)" (Traditional (Generate Executable File)).
- 编译器位置** (Compiler Location): A text input field with a browse button "...".
- 解释器位置** (Interpreter Location): A text input field with a browse button "...".
- 源程序扩展名** (Source File Extension): A text input field.
- 中间字节码扩展名** (Intermediate Bytecode Extension): A text input field.
- 默认编译器参数** (Default Compiler Arguments): A text input field.
- 默认解释器参数** (Default Interpreter Arguments): A text input field.

At the bottom, there are three buttons: "< 上一步(B)" (Previous Step), "下一步(N) >" (Next Step), and "取消" (Cancel).

编译器名称 编译器在列表中显示的名称。

编译器类型 总共有三种类型可选，括号里给出了相关解释。C++、Java 和 Python 分别是三种类型的典型代表。

编译器位置 如果编译器的类型是传统型或需要编译的解释型，这里就要选择编译器的位置。对于传统型，编译器会将源代码直接转换成机器代码，而解释型的编译器会生成中间字节码。

解释器位置 如果编译器的类型是解释型，就要选择解释器的位置。解释器用于执行中间字节码或直接解释执行源代码。

源程序扩展名 用于判断哪些扩展名的源程序使用这个编译器编译，如果有多个扩展名请用 ';' 隔开。

中间字节码扩展名 解释型编译器生成的中间字节码的扩展名，例如 Java 的中间字节码扩展名为 `.class`。

默认编译器参数 编译时传递给编译器的参数，其中用 `%s` 表示不带扩展名的源程序文件名，`%s.*` 表示带扩展名的源程序文件名。例如 g++ 的编译参数为

```
-o %s %s.* -lm
```

默认解释器参数 运行解释器时传递的参数，表示的方法同编译器参数。

向导完成后，我们再回到编译器管理的选项卡。

如果要删除当前选择编译器，可以按右边的减号。

右边的上下箭头是用来调整编译器优先级的。编译器的优先级是指：如果选手对于同一道题提交了多种扩展名的源程序，排在前面的编译器会被先考虑，同时在源程序扩展名中设置排在前面的扩展名会被先考虑。

点击“高级选项”后可以进入下图的对话框来修改已有编译器的配置。



由于不同语言执行效率有差异，因此可以放宽特定语言的时间限制或空间限制，也就是将时间限制或空间限制乘上一个实数。你也可以选择直接取消空间限制。

每个编译器都可以有多个配置，不同配置的编译参数或解释器运行参数不同，一般用于选择不同的优化开关。

点击环境变量按钮可以设置编译器和程序运行时额外设置的环境变量，一般用于保证运行所需的动态链接库文件能被找到。

5.3 视觉设置

点击上方的“视觉”选项卡就能进入视觉配置。

这个配置用于改善 LemonLime 的美观度以及提升定制性。



满分 HSL 颜色 满分时的背景颜色，用 HSL 标准表示。从左往右第 1 个框是 H 值，范围为 0 到 720。第 2 个框是 S 值，范围为 0 到 100。第 3 个框是 L 值，范围为 0 到 100。

背景 HSL 颜色 背景的颜色。不是满分的颜色为从背景 HSL 颜色到满分 HSL 颜色的根据得分与满分的比例的平滑变换。

无文件 HSL 颜色 当某个选手某个题目找不到文件的时候的背景颜色。

编译错误 HSL 颜色 当某个选手某个题编译错误的时候的背景颜色。

总分颜色补正 总分一栏的颜色会对应地加上总分颜色补正的值。

总分颜色变化倍率 总分一栏的颜色的变化倍率会对应地乘上总分颜色变化倍率的值。

启动横幅毫秒数 启动时横幅显示的毫秒数。0 表示关闭。

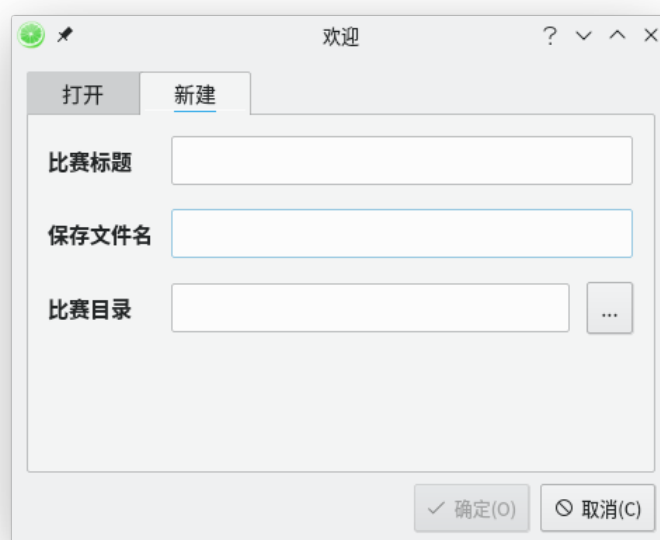
重置到默认 将以上所有值重置到默认。

你可以恰当地调整它们的值，来实现一些精彩的效果，比如实现暗色主题。

6 比赛

运行软件后会弹出“欢迎”对话框，可以选择打开已有的比赛或者新建比赛。如果关闭了这个对话框，也可以在文件菜单中选择打开或新建比赛。

新建比赛的对话框如下图所示：



比赛标题 用来显示在软件标题栏上的比赛标题。

保存文件名 保存比赛文件使用的文件名，这里输入的不需要附带扩展名。

比赛目录 比赛相关文件的存储目录。

点击确定后，软件会在制定的比赛目录下创建 **data** 和 **source** 目录，同时还有一个扩展名为 **cdf** 的文件名用来保存试题、选手信息。

其中 **data** 下用来存放试题的数据、自定义的校验器、交互题的交互库、接口实现等文件，**source** 目录下存放每个选手的源程序或答案文件。

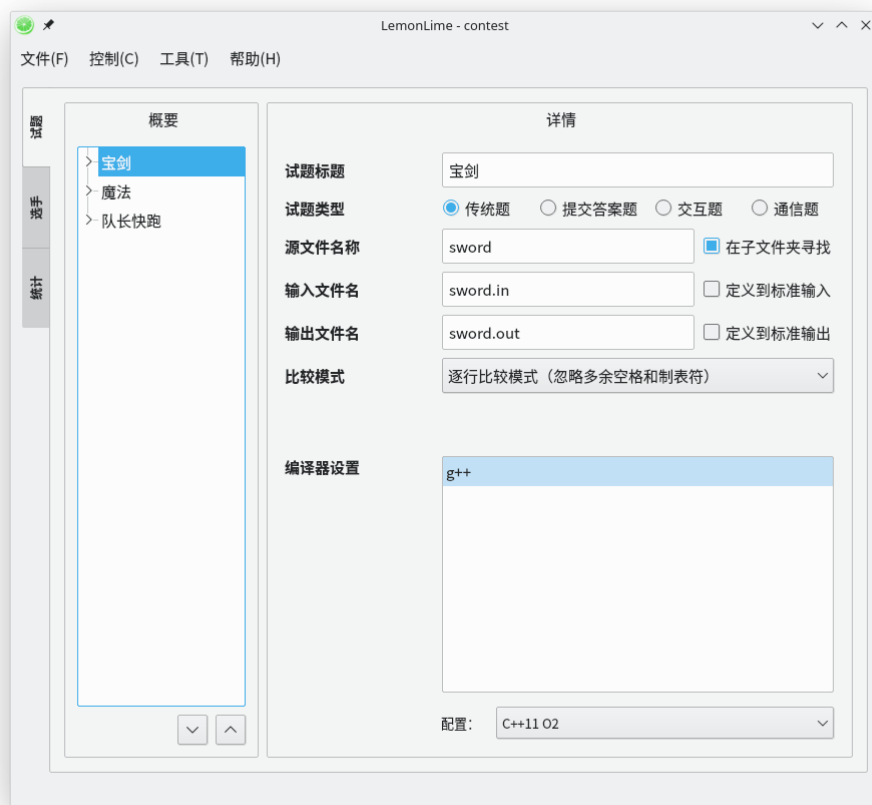
source 目录下的每个文件夹代表一位选手，文件夹的名称为选手的名称。每位选手的文件夹是否需要为每个题目建立子文件夹可以在题目页面设置，见以下内容。

你可以使用文件菜单栏中的“打开当前比赛目录”来打开当前比赛的目录。

你还可以使用文件菜单栏中的“更改比赛标题”来更改当前比赛的标题。

6.1 添加新试题

建立好比赛后，在左边按鼠标右键就可以添加新的试题。然后在右边设置试题相关的信息。



概要下的一对箭头 点击它们可以改变当前题目在列表的位置。

试题标题 试题在列表中显示的名称。

试题类型 试题的类型，目前可用的选择有传统题、提交答案题、交互题和通信题，其中交互型只支持 NOI 风格的 C++ 语言交互。

源文件名称 源程序的文件名，注意不要带扩展名。并不一定要和试题标题相同。

在子文件夹中寻找 若勾选，选手目录中需要为每个题单独建一个以源文件名称命名的文件夹，将源程序放在文件夹内；否则只需要将源程序放在选手目录下即可。

输入、输出文件名 选手程序使用的输入输出文件名。

定义到标准输入、输出 若勾选，则选手程序（交互题和通信题则是选手程序和接口文件编译出来的程序）会从标准输入读入数据（或向标准输出输出数据），不使用文件 IO。

比较模式 比较选手输出和标准输出的方式，目前有五种方式：逐行比较模式、忽略多余空格和制表符的逐行比较模式（默认）、外部工具模式、实数比较模式和自定义校验器。

逐行比较模式会一行一行比较选手的输出和标准输出是否相同，不同系统平台的换行符不同不会产生影响。

逐行比较模式中也可以选择忽略多余的空格和制表符（这也是推荐的）。

外部工具模式会调用 Linux 下的 `diff` 命令进行比较，但是小心 Windows 下可能没有 `diff`。

实数比较模式会注意读取选手输出和标准输出中的每一个实数，分别比较误差（绝对误差和相对误差）是否在允许范围内，并且判断 `nan` 和 `inf`。

自定义校验器需要选择一个可执行文件作为校验器，具体的说明请参见下一个章节。

编译器设置 为每个编译器选择配置，也就是选择相应的编译参数，默认会选择 `default` 配置。

选手答案文件扩展名 这个只在提交答案题可见。对于提交答案题，选手提交的答案文件中，每个文件会和输入文件中去除扩展名后文件名一样的那个配对，这里可以设置选手提交的答案文件的扩展名，默认为 `out`。

交互库路径 这个只在交互题可见。对于交互题，使用的交互库路径，通常为 `.h` 或 `.hpp` 文件。

交互库名称 这个只在交互题可见。指选手需要引用的头文件名称。

接口实现 (grader) 路径 这个只在交互题可见。一个实现交互库中的接口的文件。

源文件列表 这个只在通信题可见。这个应该包含选手的所有要写的程序。可以通过右边的按钮来增删内容。

接口文件列表 这个只在通信题可见。这个应该包含所有要用到的接口文件。可以通过右边的按钮来增删内容。注意这里的路径以 `data` 为根。

路径 / 文件名 这个只在通信题可见。如果点击以上两项的“添加”，则会把这一栏的内容添加进对应项目。

6.2 自定义校验器 (SPJ) 说明

自定义校验器需要为一个可执行文件，评测软件通过将一些参数传给校验器，使得校验器获得输出、输出文件等信息，然后将得分的结果写入指定文件中。

评测软件会向校验器传入六个参数，按照顺序分别表示标准输入文件、选手输出文件、标准输入文件、本测试点满分、分数输出文件、额外信息文件。

其中分数输出文件必须创建，需要向其中写入一个非负整数表示得分。

额外信息文件可以不创建，如果创建了可以写入任何信息，这些信息会显示在结果中。

6.2.1 校验器实例

下面是一个例子。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  ifstream fin, fout, fstd;
5  ofstream fscore, freport;
6
7  inline void quit(int score, const char *msg)
8  {
9      fscore << score;
10     freport << msg;
11     fin.close();
12     fout.close();
13     fstd.close();
14     fscore.close();
15     freport.close();
16     exit(0);
17 }
18
19 int main(int argc, char **argv)
20 {
21     fin.open(argv[1]);
22     fout.open(argv[2]);
23     fstd.open(argv[3]);
24     int fullScore = atoi(argv[4]);
25     fscore.open(argv[5]);
```

```

26     freport.open(argv[6]);
27     string ps, js;
28     fout >> ps;
29     fstd >> js;
30     int P = js.length(), Q = ps.length(), K = 0;
31     for (int i = 0; i < min(P, Q); i++)
32         if (js[i] == ps[i])
33             K++;
34     if (Q > P)
35         K -= Q - P;
36     if (K >= P)
37     {
38         quit(fullScore, "Success");
39         return 0;
40     }
41     double ratio = 1.00 * K / P;
42     if (ratio >= 0.6)
43     {
44         quit(0.4 * fullScore, "Great: 40%!");
45         return 0;
46     }
47
48     quit(0, "Failed: @_????????");
49
50     return 0;
51 }

```

6.2.2 使用 testlib

testlib 的使用说明可以在 [OI Wiki](#) 上见到。

Lemon 所需的修改版 testlib 已经附在 LemonLime 源代码仓库的 assets 中，由 [PinkRabbit](#) 维护，地址在 [GitPinkRabbit/Testlib-for-Lemons](#)。

注册 checker 只需要执行一句 `registerLemonChecker()` 即可。

把上面的 spj 重写后如下所示：

```

1  #include "testlib_for_lemons.h"
2  #include <bits/stdc++.h>

```

```
3
4 using namespace std;
5
6 int main(int argc, char **argv)
7 {
8     registerLemonChecker(argc, argv);
9     int fullScore = perfectScore;
10    string ps = ouf.readToken();
11    string js = ans.readToken();
12    int P = js.length(), Q = ps.length(), K = 0;
13    for (int i = 0; i < min(P, Q); i++)
14        if (js[i] == ps[i])
15            K++;
16    if (Q > P)
17        K -= Q - P;
18    if (K >= P)
19    {
20        printf(_ok, "Success");
21        return 0;
22    }
23    double ratio = 1.00 * K / P;
24    if (ratio >= 0.6)
25    {
26        printf(0.4 * fullScore, "Great: 40%%!");
27        return 0;
28    }
29    printf(_wa, "Failed: @_????????");
30
31    return 0;
32 }
```

6.3 关于非传统型试题

本节介绍了非传统型试题与传统题的区别以及测评逻辑。

6.3.1 提交答案题

提交答案题中，选手不需要提交源程序，只需要提交答案文件。

所以测评中，不需要进行源程序的编译、运行，直接对答案进行判断即可。

选手提交的答案文件名应与测试点配置中的**输入文件**相同（不包括扩展名），扩展名与题目配置的答案文件扩展名相同，软件将对选手提交的文件与测试点配置的答案文件以配置的比较模式进行比较。

6.3.2 交互题

只支持 NOI 风格的 C++ 语言交互。

编译时将交互库拷贝至编译临时目录下并命名为交互库名称，将接口实现文件拷贝至目录下，进行双文件编译。

对于交互库全部写在一个库文件中的题目（不推荐，选手可能会通过扫内存等方式获得信息），可以创建一个空的接口实现文件 `grader.cpp` 完成配置，不影响编译。

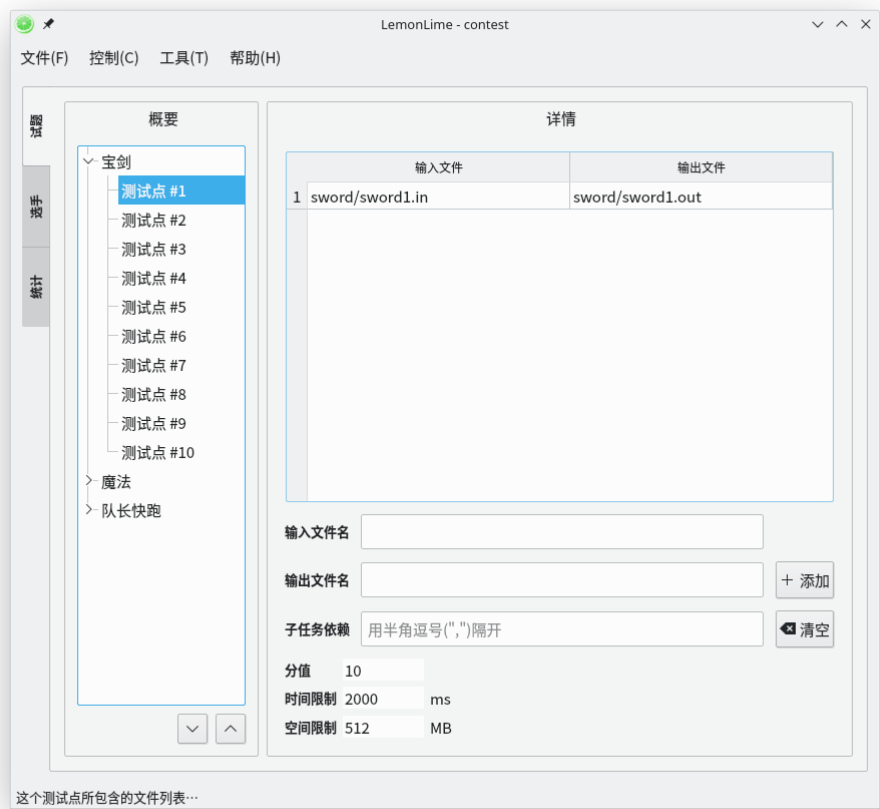
6.3.3 通信题

编译时将对应的所有文件拷贝至编译临时目录下进行多文件编译。

6.4 添加新测试点

在左边选中一道试题后，右键鼠标出现菜单，选择“添加测试点”即可添加一个新的测试点，右边会变成测试点设置界面。

在输入文件名和输出文件名中输入内容后，点击“添加”按钮即可添加一组测试数据。这里的输入输出文件必须在 `data` 目录下，并且只要输入 `data` 目录内的相对路径即可，如下图所示：



一个测试点可以包含多组输入输出，最终一个测试点的得分为该测试点包含的所有测试数据中最低的得分。

如果想要使用子任务依赖的话，对于测试点 i ，请保证输入的子任务编号在 $[1, i - 1]$ 之间，多个依赖项之间用半角逗号（,）隔开。子任务依赖的意思是这个测试点不会在被依赖的测试点中有错误（不是答案正确，且不是答案部分正确）的情况下测试。注意如果想清空的话，必须点击右边的“清空”按钮。

如果要编辑输入输出文件名，直接在表格相应位置双击即可修改。
选中一行或多行后按 **Delete** 键，即可删除对应的输入输出文件。
在下面可以设置本测试点的分值、时间限制和空间限制。可以设置的最大分值为 10000000，最大时间限制为 86400000 ms（即 1 天），最大空间限制为 16777216 MB。

6.5 批量添加测试点

在左边选中一道试题后右键鼠标出现菜单，选择“添加多组测试点…”后会弹出一个向导，用来批量添加测试点。

向导的第一步是设置每个测试点的分值、时间限制和空间限制，当然也可以添加完成后在编辑页面更改。

第二步是设置如何匹配输入、输出文件名，这里需要会使用简单的正则表达式。

输入、输出文件格式中可以使用 `<1>`, `<2>`, ..., `<9>` 来表示一个正则表达式，然后在下面按右边的加号可以先见这样一个参数并指定参数代表的正则表达式。需要注意的是在输入文件格式和输出文件格式中，每个参数只能出现一次。

对于所有参数匹配内容都相同的文件，会作为一组输入输出。

对于打钩的参数匹配内容都相同的输入输出，会放在同一个测试点中。

以一道捆绑测试的试题为例，在 `data` 下的 `sword` 文件夹里，有 `sword(x)-(y).in/out` 文件代表第 x 个子任务的第 y 个测试点的输入输出文件（ x 和 y 都是数字），那么匹配参数可以按照下图：

添加多组测试点

步骤 2/3: 为输入输出文件指定格式。你可以使用像`<1>`, `<2>`...这样的参数来表示一个正则表达式。打钩的表达式匹配内容相同的数据点将会在一个 Subtask 中。

输入文件格式: `sword/sword<1>-<2>.in`

输出文件格式: `sword/sword<1>-<2>.out`

参数	正则表达式
<input checked="" type="checkbox"/> <code><1></code>	<code>\d*</code>
<input type="checkbox"/> <code><2></code>	<code>\d*</code>

< 上一步(B) 下一步(N) > 取消

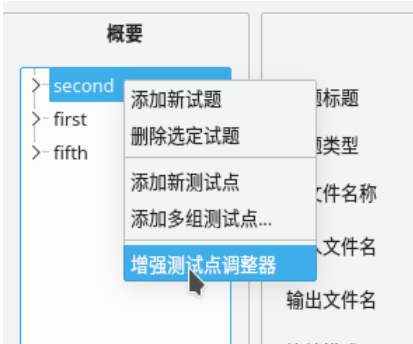
注意：Windows 下文件夹分隔符为 `\`，Linux 下文件夹分隔符为 `/`。

最后一步是预览结果，如果发现结果和预期的不同可以回到上一步修改参数。

6.6 增强测试点调整器

增强测试点调整器是模仿项目 CCR-Plus 的一个特色功能。你可以用它来非常方便地做一些以前未曾设想过的操作。

在试题栏的概要框中，右键题目即可出现进入通道。



进入之后如下图所示：



新测试点 新增一个测试点。

新数据 在当前选中的一个测试点中新增一组测试数据。

修改选中 修改选中的测试点或测试数据。你能修改的条目和你的选择对象有关。

上移 上移当前选中的测试点或者同一个测试点的测试数据。

下移 下移当前选中的测试点或者同一个测试点的测试数据。

合并 将当前选中的测试点合并。分数将是被选中的测试点的分数之和，而其它限制将继承选中的第一个测试点。

拆分 将当前选中的测试点拆分成只有一组测试数据的测试点。分数将会在各个测试点均分。

移除 移除当前选中的测试点和测试数据。

确定 / 取消 只有选择确定之后更改才会被应用。

注意：所有造成测试点数量更改的操作都会清空所有测试点的子任务依赖。
这个功能还在测试阶段，请小心使用！

6.7 自动添加试题

为了简化试题、测试数据的添加过程，Lemon 作者设计了自动添加试题功能。对于一道试题，如果希望能够自动添加，请在 `data` 目录中为这个试题创建一个文件夹，将相对应的数据文件放入文件夹中。然后再控制菜单中选择“自动添加试题”，会出现一个对话框，给出找到的试题。

可以设置每道试题所有测试点的总分值、所有测试点的时间限制和空间限制，确定后就会自动添加相应的试题。

自动添加的试题默认作为传统型试题，添加后的试题标题和源程序名称都是对应试题的文件夹名称，输入输出文件分别再加上扩展名 `in` 和 `out`。

数据文件的匹配方法是：根据设置中设定的输入输出文件扩展名，选出相应的文件，如果在设置中输入或输出文件扩展名为空，会自动将输入文件扩展名设置为 `in`，输出文件扩展名设置为 `out;ans`。

然后对于除扩展名外文件名相同的文件会被作为一组输入输出，并为这一组输入输出创建一个测试点，注意文件名的大小写是敏感的。

用“自动添加试题”功能添加的试题，每个测试点只会有一组输入输出。

每个测试点的分值会用设置的总分除以测试点个数，如果不能分配均匀的话，那么余下的点数会分配给后面的测试点。

请尽量保证输入输出文件能够按上述方法唯一配对，否则产生的结果不可预料。

其中可以设置的试题总分值最大为 10000000，最大时间限制为 86400000 ms（即 1 天），最大空间限制为 16777216 MB。

7 测试

点击“选手”选项卡，点击下面的“刷新”按钮后，就会根据 `source` 目录下的文件夹添加列表中不存在的选手，并从列表中删除已经从 `source` 目录中删除的选手。

然后单击“测试全部”按钮就能开始测试。注意选手提交的程序名在 Linux 下是大小写敏感的。

单击“测试未测试”按钮可以测试所有还没有测试的元素（即某个选手的某个题目）。

选中一些元素后按“测试选中”按钮可以仅测试选中的部分。如果选中了某选手的排名、名称、总分、总用时、测试时间的其中一个，那么将默认测试这个选手的所有题目。

在控制菜单中还提供一键测试所有未找到文件的或者所有编译出错的记录。

按 `Delete` 键或右键点击删除可以删除选中的选手。

测试完成后通过在表头上单击可以按照相应的项目排序。

双击一名选手可以查看详细的测试结果并对该选手的某一题进行重测。

测试过程中会有 Subtask Skip，即对于一个测试点如果已经确定该测试点会得 0 分，则跳过这个测试点剩下的所有数据，可以大大减少捆绑测试的题目的测试时间。

当然，LemonLime 还提供强制跳过某个题目的测试的功能。请不要在正式场合中使用。

7.1 测试情况对话框



这个对话框会实时显示当前评测的情况。

各种评测结果的字体颜色各不相同。

如果选手在某一个测试点“答案正确”或者“答案部分正确”，那么会显示它的得分、用时和所使用的空间。

进度条上会显示当前进度和总时间限制。

7.2 颜色

不同的选手的不同的题目会根据得分和评测状态呈现出不同的颜色。

如果想改变颜色，可以到“设置”中改变。

7.3 整理文件

为每个选手的每一个文件创建它的子文件夹内的文件和子文件夹外的文件，无论它们以前是在子文件夹内还是子文件夹外，并且删除大部分无用文件。

如果选手原本子文件夹内外都有文件，子文件夹外的那一个会被覆盖。

注意这个功能对于提交答案题的支持还不够完善。最好在整理文件之前备份一份，以防不测。

7.4 跳过这一题

当你肯定某个选手是卡评测或者其它原因，或者这个题的评测结果无关紧要（比如自己自娱自乐的时候），那么可以点击“跳过这一题”。

这会打断当前评测的题目，并跳过这道题之后的所有测试点。

跳过的测试点会显示超过时间限制。

8 导出成绩

在“控制”菜单中选择“导出成绩”可以将结果导出成 HTML 文档或表格文件。

推荐使用 HTML 格式，可以导出完整的结果信息，导出成表格的话只能导出选手总分和每道题的得分。

表格有两种格式可以选择：csv 格式和 xls 格式（仅 Windows 可用并需要安装 Excel，并且需编译时使用 `-DENABLE_XLS_EXPORT=ON` 参数启用）。注意：xls 格式的导出将有可能在将来的版本移除

csv 格式是逗号分隔符，多数表格编辑软件都能查看。xls 格式的导出需要利用 ActiveX 调用 Excel，写入速度非常慢，除非特别需要，否则不推荐使用。

在导出为 HTML 格式的时候，后缀（.html 和 .htm）会影响内容。其中 .html 的内容会更加丰富，而 .htm 只提供基本内容，并且文件大小下降很多。

导出为 .html 的时候可以选择是以当前的颜色风格导出还是以默认的颜色风格导出。如果你使用的是暗色主题，可能需要额外注意这一点。

9 比赛统计

点击“统计”选项卡即可查看这场比赛的统计信息。

9.1 总览

这个部分显示了这场比赛的总体情况。



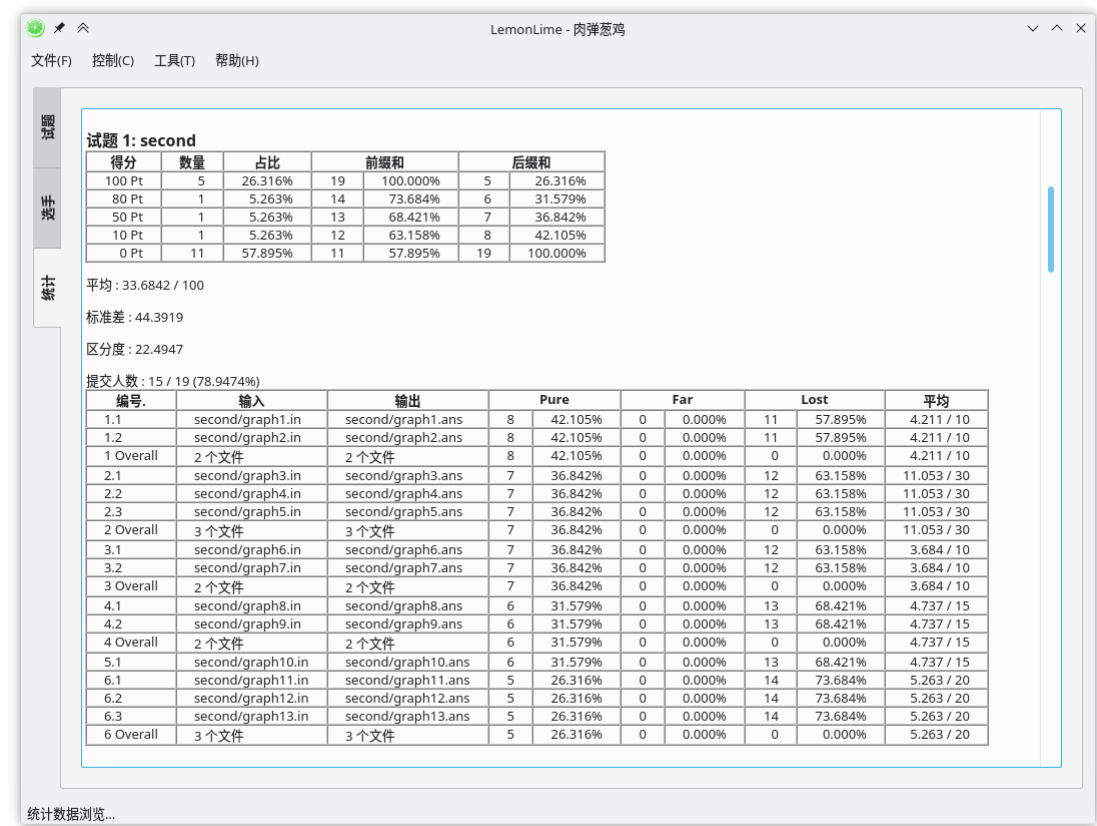
你可以获得的信息有：

- 得到某个分数的选手人数以及比例，以及它的前缀和和后缀和。
- 这场比赛的平均分、分数的标准差和区分度。

其中，区分度是一个衡量本场比赛或某个题目的区分度的值。这个值仅供娱乐。

9.2 题目

每一个题目都有属于自己的一部分。这个部分显示了某个题目的情况。



你可以获得的信息有：

- 得到某个分数的选手人数以及比例，以及它的前缀和和后缀和。
- 某个测试点的通过情况。
- 这个题目的平均分、分数的标准差、区分度和提交人数。

9.3 导出统计信息

你可以使用控制菜单栏中的“导出统计信息”来导出当前比赛的导出统计信息到一个 HTML 文件。

这个 HTML 文件的内容和 LemonLime 中的内容将完全一致。

10 常见问题及回答

在使用中出现任何问题，可以在 [Github 仓库](#) 中的 Issue 提出。
此节内容长期更新，更多问题欢迎提出。

10.1 测评时使用更多栈空间

Windows 平台下可以在 g++ 编译时添加 `-Wl,--stack=2147483647` 命令来开启约 2048 MB 栈空间。

Linux 平台下的栈空间限制和题目的内存限制相同。