形式语言与自动机 第三次实验 捕获分组与复杂正则表达式 实验文档

实验概述

本实验需要大家以编程的方式完成，目标是在第二次实验写出的正则表达式执行器的基础上，加入更丰富的功能，使你的正则表达式执行器更强大。-
要求实现的功能有：

- 输出捕获分组
- 支持处理多行文本
- 支持两种正则表达式flag: multiline(`m`)，single line(`s`)
- 支持anchor字符：`^` `$` `\b` `\B`
- 支持rangeQuantifier如`{2,5}`
- 支持matchAll和replaceAll，replace中支持分组引用如`$1`

编程语言

本实验需要大家在**C++**或**Python**语言中任选一种进行完成。
对于每种语言，我们都提供了一套SDK（半完成的代码框架），大家只需按要求完成相应的函数即可。

关于外部依赖，在全部的实验中，**不允许大家使用任何的外部依赖。**
即C++语言只能使用标准库，Python语言只能使用系统库，不可以引入任何第三方库（无论是以源代码复制、pip、cmake或是任何其他形式都不可以）。此外，标准库或系统库中与正则表达式有关的库（如C++的std::regex，Python的re）也不可以使用。

实验框架内容概述

你收到的实验框架中包含的内容和含义描述如下表：

目录名	描述
cpp	C++语言的编程框架。具体的用法参见实验具体说明的C++语言部分。
python	Python语言的编程框架。具体的用法参见实验具体说明的Python语言部分。
antlr	包含正则表达式的文法定义`regex.g4`和ANTLR的jar文件等。具体的用法参见实验具体说明的ANTLR部分。
cases	存放测试样例的文件夹。每个测试样例是一个txt文件。
judge	内含不同平台上的评测器的可执行文件。命名均为judge-操作系统-架构，请自行选择合适的评测器运行。如果没有你合适的平台的可执行文件，请直接联系助教。

实验要求

本次实验的评分包括两个部分：

- 对第三次实验的全自动评测，着重考察第三次实验所要求新增的功能。

    - 自动评测的含义是：你的程序将被多次调用、输入多个不同的测试样例进行测试。对每个测试样例，若你的程序在规定时限内正常返回并给出正确的结果，则该测例得分，否则不得分。
- **对全部三次实验**的代码和文档的评价，依据是你在本次实验中提交的全部代码和文档。
    - 本次实验**需要提交文档**，且文档内容**应当针对全部三次实验**的内容。
    - 文档内容应当至少包括：编程语言与环境、主要实现思路、实现过程中的重难点问题及你的解决方法等。要求简明且深刻，重在个人的体验和感悟。
    - 文档建议**不超过1000字**。过长的文档不会让你的评分更高，反而很可能由于冗长而失去重点。

提交方法

请在网络学堂的DDL前，将你的作业提交到**网络学堂**。提交的内容为一个压缩包（建议最好是`.zip`格式），内容包含：

- 若你是使用C++语言完成，请提交：
    - `cpp`文件夹
        - 请确保删除了`cpp`文件夹内的中间产物文件夹如`build`、`cmake-build-*`等。
        - 请勿删除`CMakeLists.txt`、`lib`文件夹和`parser`文件夹，否则你的代码将无法编译！
    - 在你编程的平台上编译出的可执行文件。Windows请命名为`regex.exe`，Mac和Linux请命名为`regex`（即CMake的默认编译输出名）
        - 此项用于在助教无法亲自编译你的代码的情况下作为参考。如果助教能够成功编译你的代码，则不会使用你提交的可执行文件，此种情况下即使你未提交可执行文件也不会扣分。
    - 一个简短的TXT文件，命名为`README.txt`，说明你的编程平台（如操作系统、架构、编译器等）。如果有其他想给助教留言的也可以写在这里。
    - 请注意不要提交`python`文件夹，否则可能造成评测器混淆你的编程语言，严重影响评测分数。
- 若你是使用Python语言完成，请提交：
    - `python`文件夹。
        - 请勿删除`parser`文件夹。
    - 一个简短的TXT文件，命名为`README.txt`，说明你的编程平台（如操作系统、架构、Python版本等）。如果有其他想给助教留言的也可以写在这里。
    - 请注意不要提交`cpp`文件夹，否则可能造成评测器混淆你的编程语言，严重影响评测分数。
- **此外，请提交你的文档，要求pdf格式**，建议命名`实验文档.pdf`。
- 此外，虽然我们要求原则上不得修改文法文件，但如果你确实由于某些原因修改了文法，则请额外提交`regex.g4`文件备查。

评分规则

本次实验中，自动评测部分占据实验部分总成绩的30%，综合评价部分占据实验部分总成绩的20%。

- 自动评测部分的评分规则为：
    - 公开测例 60% (20个，每个测例分值均等)
        - 已包含在本次下发的实验框架中，同时提供了自我测评用的评测器。
        - **使用评测器得到的自测得分仅供参考**，不作为得分的直接依据。
    - 隐藏测例 40% (每个测例分值均等)

        - 不会公开给同学。将在DDL后由助教进行测试。
    - 所有的测例，无论是公开还是隐藏，均将**由助教在DDL后统一运行你的代码进行测试并计算得分**。得分以助教评测的结果为准。
- 减分项：如你存在下列问题，可能会被额外进行惩罚性的减分。
    - 抄袭：**本实验和其他的所有实验均严禁抄袭**。抄袭者最严重将被处以所有实验全部0分的惩罚。不能给出合理解释的代码高度雷同也被视为抄袭。
    - 攻击评测机：禁止用任何方式攻击评测机，包括但不限于尝试访问、修改与自己的实验无关的文件、执行恶意代码、尝试提权等行为。违反者视情节，最严重将被处以所有实验全部0分的惩罚，如涉及违纪违法还将上报学校等有关部门处理。
    - 使用非正常手段通过测例：包括但不限于针对特定的输入直接匹配输出，通过联网、调用评测机等手段从外部来源获取答案等。违反者将被扣除所有以非正常手段通过的测例的得分。
    - 未按要求提交：请严格按照上面的提交要求提交作业。
        - 若未按要求提交引起评测器无法自动评测分数的（如同时提交cpp和python文件夹等行为），将被至多扣除本次实验总分的20%。
        - 其他情况，至多扣除本次实验总分的10%。
- 综合评价部分的评分规则为：
    - 文档评分：50%（即实验总成绩的10%）
        - 文档要求详见实验要求一节。
    - 代码质量评分：50%（即实验总成绩的10%）
        - 评价内容包括但不限于：代码结构合理性，编程规范性，代码可读性，符合最佳实践，注释清晰等。
        - 仅针对本次提交的代码进行评价。前两次提交的代码不会做任何评价，也并不要求你本次提交的代码在前两次实验的部分需要与此前提交的完全相同。

迟交政策

由于涉及登记成绩的问题，本实验采用如下的迟交政策：

- 毕业年级同学的DDL按照单独的规定，已由助教通知本人。
- 其他同学，DDL为2023年6月27日23:59。
- 在此基础上，**每迟交12小时**，分数扣减5%。
    - 分数扣减同时作用于自动代码评分和文档评分两部分上。
- **至多允许迟交36小时。**超过2023年6月29日11:59后，**将不再接受任何补交。**

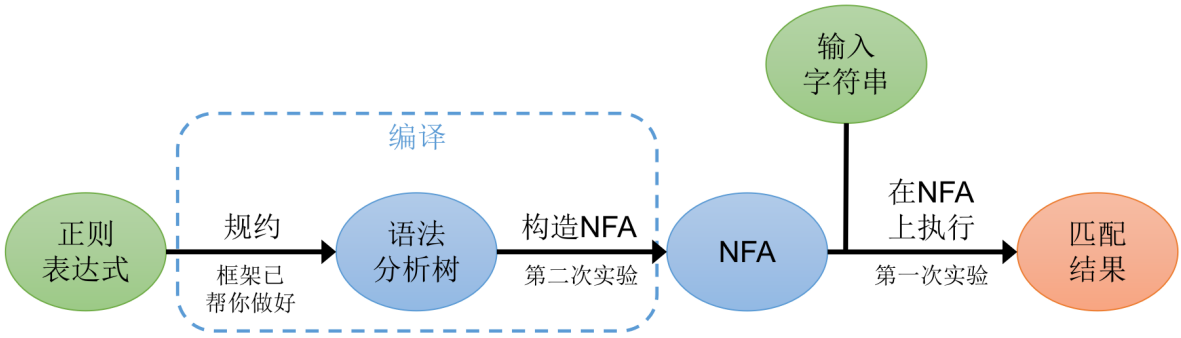
其他

- 第三次实验中：
    - 保证无论是正则表达式(pattern)字符串，还是待匹配文本字符串中，都只包含ASCII字符（字节值0~127），且不会包含NULL字符`\0`。
    - 额外地，pattern字符串中保证不会含有换行符(即pattern只有一行)，但**待匹配文本字符串中可能含有换行符**(即文本可能具有多行)。
- 程序"正常返回"的定义：你的程序正常结束执行，进程返回码为0。
- 助教评分时使用的评测环境：
    - Ubuntu 22.04 LTS (in docker)
    - Intel i7-12700K (5.0GHz)
    - Python 3.11

- 关于程序限时：本实验给出的限时可以说是非常宽松的。
  - 对于C++语言，每个测例限时5s；对于Python语言，每个测例限时10s。
  - 限时一律以程序在助教的评测机上的用时为准，均包括IO时间。
  - 超时的测例在评测器中会分类在“发生异常”类别中。
  - 随本次实验下发的自测用评测器现已实现超时杀死进程的机制。
- 关于转义字符：
  - 转义字符实际上包括两种情况。第二次实验中，我们为对此作出明确说明，测试样例中也未包含覆盖第一种情况的例子。因此我们在后来补充宣布第二次实验必定不含`\r\n\f\t\v`。
  - **但本次实验中可能包含它们**。请阅读转义字符小节查看具体的解释。

实验具体说明

为了实现一个正则表达式引擎(Regular Expression Engine)，我们整体的思路如下图所示：



第二次实验中，我们已经完成了这个思路图中的全部环节，并且构建出了一个支持部分语法的正则表达式执行器。
本次实验，我们需要在第二次实验的代码的基础上进行修改，以支持更多的特性。

本文档中的实验思路仅为纯文本格式的一部分提示，建议你到网络学堂下载实验讲解PPT结合阅读，PPT中有更多的讲解和图片例子等。

捕获分组

分组是正则表达式中用括号括起来的一部分，事实上，在正则表达式中，括号可以同时起到两种作用：调整运算的优先级，和单独提取出分组内的捕获内容。
在用户要求的情况下，正则表达式引擎除了返回整个表达式的匹配结果外，还需要返回结果中每个分组所对应的内容。在第二次实验中我们未要求这个特性，但本次实验中，你需要实现它。

例如对pattern `www\.(\w+)\.(\w+)`、输入串 `https://www.baidu.com/` 时，第二次实验中我们是仅给出匹配结果 `www.baidu.com`。
在本次实验中，除了完整匹配结果，我们还要求大家输出分组结果，在本例中有两个分组`baidu`和`com`。

当一个pattern中有多个分组时，应当对分组进行编号，编号的方法是按照**左括号出现的顺序**。例如，`(a(bc)+)(de)+`，第1分组是`(a(bc)+)`，第2分组是`(bc)+`（因为前者的左括号在后者前面），第3分组是`(de)+`。此外，我们额外规定**完整匹配结果称为第0分组**。
处理分组编号的过程应当并不复杂，只要你恰当地处理各类节点遍历子节点的顺序，就很容易可以保证编号小的分组先被遍历到。

分组分为捕获分组和**非捕获分组**。捕获分组就是普通的一对括号，具有分组编号，需要返回其分组内容；而非捕获分组在第一个括号后额外有一组`?:`，例如`(?:ab+)`。
非捕获分组只起到调整运算优先级的作用，正则表达式引擎**不应对其编号**、也不应返回其分组内容。

`match`函数返回的内容是一个字符串数组，在第二次实验中你永远只需要返回一个元素，但本次实验中，你应当通过该数组返回所有的分组的匹配结果，从**第0分组(即完整匹配结果)**到最后一个分组。
例如对上面的例子，match函数的返回值是`["www.baidu.com", "baidu", "com"]`。

捕获分组的输出结果就是完整匹配结果中的一段子串。而完整匹配结果是NFA执行的一条接受路径，因此，捕获分组本质就是该路径上的一段子区间内的内容。
在处理group类型的节点时，我们可以在构造自动机的同时，就在自动机的状态上**加上一些标记**，用于标记每个分组的开始和结束。这样，当我们执行NFA拿到接受路径后，就可以根据该路径所经过的状态，提取出每一对标记之间对应的字符串内容，即为分组的结果。

此外，还有如下规则：

- 当某个分组没有任何匹配结果、接受路径上实际根本没经过该分组时(常见于多个表达式取或，某个分支内的分组不是或运算实际选择的分支的情况)，该分组输出结果为空串`""`。函数返回值中绝不能省略结果为空分组，否则分组序号的对应关系会发生混乱。
- 当一个分组本身被经过多次时，应当返回**最后一次**的结果。
- 如pattern`(a\w)+`，输入串`abacad`，则第1分组的结果应为`ad`。

anchor字符

anchor字符是一类特殊的元字符，它不匹配一个具体的字符，而是匹配一个位置。
具体而言，它有两个特点：

- 匹配时，不止看当前位置字符是什么，还要看前面的一个字符是什么。
- 匹配成功后，不消耗字符。

我们要求支持的anchor字符包括(下列描述的含义指在无flag的情况下)：

- `^`：匹配字符串的开头
- `$`：匹配字符串的结尾
  - 例如，对字符串`abcde`，pattern`bcd`可以匹配出结果，但pattern`^bcd$`不能匹配出结果。
- `\b`：匹配单词边界
  - 单词边界定义为，按照`\w`与`\W`分类，当前位置前后的两个字符不同类。当当前位置前或后没有字符时，视为是`\W`类。
  - 例如，对字符串`abcde`，pattern`abc`可以匹配出结果，但pattern`abc\b`不能，因为匹配完字母`c`后，`c`和`d`之间属于非单词边界。
- `\B`：匹配非单词边界。与`\b`刚好相反，它要求当前位置前后的两个字符同类。

代码实现中，建议把anchor字符视为一种**附条件的ε-转移**。一般的ε-转移可以看作是无条件的，一定可以产生转移到下一个状态的分支；而anchor字符对应的转移只在特定的条件下进行，例如当前位置是否是开头结尾/单词边界。
在具体的代码实现上，我们建议使用此前在`EPSILON`类型的`Rule`中未用到的`by`等字段，可以用此存储该转移的条件信息，并通过修改NFA的代码实现支持。PPT中有对此更详细的伪代码描述，建议参考阅读。

根据flag，正确处理多行文本的匹配符

本次实验中，输入的字符串将可能具有多行，也就是将可能出现换行符`\r``\n`。实际上你并不需要对此做什么特殊处理，有无`\r``\n`的情况下，正则表达式运作的规则是没有变化的。

只是，当正则表达式具有flag时，你在匹配转移规则时的行为会发生变化。本实验要求支持的flag包括：

- `m`：正常情况下，`^``$`匹配的是整个字符串的开头结尾。但在有`m`flag时，`^``$`除了可以匹配整个字符串的开头结尾外，还可以匹配**一行的开头和结尾**。
- `s`：正常情况下，`.`只能匹配**除了**`\r``\n`**外的所有字符**。但在有`s`flag时，`.`也同样可以匹配`\r``\n`，真正实现匹配任何字符。

flag可能同时具有多个，通过`compile`函数的参数传入，多个flag时传入顺序没有限定，例如`ms`和`sm`是等价的，都表示同时具有上述两种flag。没有任何flag时，该参数将为空串。

matchAll和replaceAll

`match`函数要求你返回的是一维数组，是输入字符串中与正则表达式匹配的第一个结果（含分组）。而`matchAll`要求你返回的是输入字符串中与正则表达式匹配的**全部**结果，是一个二维数组，每个元素都是一个含分组的完整匹配结果。当没有任何匹配时，你应当返回一个空数组。
要求结果间没有交集，即你匹配成功一个串后，直接从前一个匹配结果尾部的下一个位置开始匹配即可。例如输入字符串`abbabbbabbbbab`和模式串`ab+ab`，两个匹配结果分别为`abbab`(0~4)和`abbbbab`(7~13)。

`replaceAll`函数的参数除了给出待匹配字符串`text`外，还给出要替换为的字符串`replacement`。它可以基于`matchAll`函数(或类似的方法)实现，将所有的匹配结果都替换为`replacement`的内容。
但有一点可能需要注意，`matchAll`函数返回的匹配的结果是字符串格式，然而你可能需要知道匹配位置的下标区间(开头、结尾在原字符串的什么位置)，才能顺利完成替换工作。

除此之外，绝大多数编程语言都支持基于分组的替换（虽然实际上没有很统一的标准、各个语言的实现不完全一致），
本实验的对此的要求如下：：

- 当你在替换串`replacement`中发现形如`$`后面跟着若干个数字的表达，如`$2`，则实际此处替换的内容应为当前位置匹配结果的第2分组。
  - 例如，对输入串`ab12ab34`、pattern`(\d+)`和替换串`c$1d`，替换的结果应为`abc12dabc34d`，直接暴力替换为`abc$1dabc$1d`是错误的结果。
  - 注意，分组序号可能不只有一位数。`$99`这种也是允许的，表示替换为第99分组。
  - 若要替换的分组在pattern中根本不存在，则约定替换为空串。例如对上例，pattern根本没有第二分组，则使用替换串`c$2d`替换的结果应为`abcdabcd`。
- 由于允许了`$1`这种替换分组，因此需要允许`$`字符转义，也就是当你发现连续的两个`$$`时，实际替换的过程中作为一个`$`处理。
- 除去以上情况，如果某个`$`后面跟的既不是数字也不是`$`，那就作为一个很普通的`$`处理即可。
- 实现上述规范的过程中，你可以使用语言标准库自带的字符串替换函数，但是只能使用基于普通字符串查找的替换函数，**绝不可以使用基于正则的替换**。违反此项规则的后果等同于使用非正常手段通过测例。

区间限定符

除去上次实验已经实现的`+``*``?`三种限定符外，本次实验还要求大家支持匹配item一定次数范围区间的限定符，
如`a{2,5}`表示匹配2-5个`a`字符，`(abc){3}`这种表示匹配连续3个`abc`，`a{3,}`表示匹配3个及以上的`a`字符。
同样的，区间限定符也是默认贪婪的，也可以通过后加`?`使其变为非贪婪。

实现思路方面，可以采取两种思路：

- `a{2,5}`，可以理解成`aaa?a?a?`，这些是在第二次实验中你已经处理过的。

    - 这种方法的在代码实现方面，由于可以复用先前的代码，可能会比较简便。缺点是构造出的状态机可能比较复杂。
- 第二种思路是，构造类似a*形式的NFA，但通过在状态转移规则中附加额外的信息，限制一些规则所可以运行的次数。
    - 例如，对`(ab)*`，我们构造的自动机通常包括匹配`ab`结束后从`ab`末尾指向`ab`开头的回环，和离开`ab`前往下一个item的出路。
    - `(ab){2,5}`则可以通过限定回环至少执行1次(在此之前不准向出路方向探索)、至多执行4次(在此之后不准走回环)来实现。
    - 更具体的，区间是否从0开始的情况，如`x{0,5}`和`x{1,5}`，实现上可能还有少许区别，对此你可以自行研究。

转义字符

本实验中，所有的转义字符都会被解析成一个`EscapedChar`类型的token。它们实际上都对应于唯一一个字符，应当按照单个字符的方式去处理和构造NFA。
具体而言，分为两种情况：

- 按照C语言等语言的惯例，用特定的转义字符匹配特定的ASCII非打印字符。
    - 本实验中要求支持此类字符共有5个：`\f` `\n` `\r` `\t` `\v`，其含义及对应ASCII码可参考 https://www.dotcpp.com/qa/17 。
- 对某个字符在正则表达式中有特殊含义的情况，转义的内容就是`\`后面的内容。
    - 如`\(`匹配的是单个`(`，`\\`匹配的是单个`\`等。
    - 按照惯例，对此类情况我们不给出具体列表，而是规定**凡是不属于前一类情况的都属于此类。**

其他提示

- 推荐使用regexr.com。这个网站可以使你在线地执行正则表达式、可视化地查看执行的结果，还能看到模式串例每个字符的含义。

代码框架公共说明——对所有的语言都适用

- cases中的测试样例均为文本文件，内含正则表达式的pattern字符串和输入字符串，文件的格式是很容易理解的，如有需要，你也可以在其基础上进行修改/编写自己的测例进行测试。
    - 如你需要自行修改/构造测试样例，**建议复制出来改动而不是在测例上直接修改**，否则如果你忘改回去了可能会造成自测分数与最终分数不一致
- 你所需要做的是完成`Regex`类的`matchAll`和`replaceAll`两个函数，修改`match`函数以返回分组匹配结果，修改`compile`函数的实现以支持新增的特性。在代码中已经使用TODO注释为你标记好。
    - 你的程序并不需要亲自从stdin中读取输入，也不需要亲自向stdout中写入结果。
    - 框架已经实现好了读取并解析文本输入，构造`Regex`类的对象并调用compile方法，再根据测例的类型调用相应的方法，最后将结果输出到stdout的逻辑。
        - 如你感兴趣，可看`main-regex.cpp`或`regex.py`文件最下面的`if __name__ == '__main__'`部分。
    - 你的程序**不得在stdout中打印任何输出**，如果确实需要打印，请**务必打印在stderr中。**
        - stdout的所有内容均会被评测器作为评测依据。
- 程序的调用方法分为两种：
    - 若不传入任何参数，则程序将从stdin中读取输入。（也是评测器评测时所用的方式）

    - 或者，你可以传入一个参数，是输入文件的路径。此时程序将改为从你指定的文件路径中读取输入。（当你想要具体的在某个测例上执行和调试时，这种方法更为方便些）
- **仔细阅读框架代码的注释！** 很多问题，包括类的含义、函数的含义、返回值的方式等，都可以在框架代码的注释中可以找到答案。
    - 框架中已经定义好了一些和函数，类内也已经定义好了一些成员变量和方法。不建议大家修改这些已经定义好的东西。
    - 但是，你可以自由地增加新的函数、类等，包括可以在已经定义好的类自由地添加新的成员变量和方法。如果你确实需要，也同样可以增加新的文件（但C++语言请注意将新增的文件加到`CMakeLists.txt`的名为`regex`的target里）。

C++语言

编译执行方法

本框架的C++语言部分使用CMake作为构建的工具。

IDE使用提示

请参见第一次实验文档。

直接在命令行中编译运行

或者，若你想直接使用命令行进行编译，方法如下：

```
cd cpp
mkdir build # 作为编译结果（可执行文件）和各类编译中间产物存储的文件夹
cd build
cmake .. # 意思是去找上级目录（此时你在build中，上级目录就是cpp）中的CMakeLists.txt文件，据此在当前目录(build)中进行中间产物的生成。这步cmake会帮你生成好一个Makefile。
cmake --build . --target regex # 执行编译
```

执行文件的方法：（注意windows平台上是regex.exe）

```
./regex # 程序会从stdin中读取数据，请自行使用输入重定向 < 、管道 | 等手段为它提供输入
./regex ../../cases/01.txt # 程序会从指定的路径读取输入。此处假定你在cpp/build文件夹下，故测试样例的相对路径应如同这个样子
```

代码结构具体描述

- `Regex`类：`regex.h` `regex.cpp`
    - 包括`Regex`类的定义。
    - 你需要实现的是`Regex::matchAll`和`Regex::replaceAll`函数，并修改之前编写过的`Regex::compile`和`Regex::match`函数。其参数和返回值含义均在注释上。请在`regex.cpp`中完成其实现。
    - 请注意，`compile`函数现在起需要处理`flags`参数了。
- 入口点文件：`main-regex.cpp`
    - 你应该不需要去管这个文件。这个仅包含`main`函数的实现，其中会构造`Regex`类的对象和调用`compile`等各种方法。

Python语言

运行方法

需要的Python版本应>=3.8，否则可能会无法运行。
首次执行代码前，务必先安装依赖:

```
pip install -r requirements.txt
```

本次实验的入口点文件为`regex.py`。

```
python regex.py # 程序会从stdin中读取数据，请自行使用输入重定向 < 、管道 | 等手段为它提供输入
python regex.py ../cases/01.txt # 程序会从指定的路径读取输入。此处假定你在python文件夹下，故测试样例的相对路径应如同这个样子
# 或者，如果你使用类Unix系统(Linux、Mac)，由于nfa文件中包含了Shebang，也可以不必输入python、直接执行regex.py
./regex.py
```

代码具体描述：`regex.py`

- 包括`Regex`类的定义。
- 你需要实现的是`Regex`类中的`matchAll`和`replaceAll`函数，并修改之前编写过的`compile`和`match`函数。其参数和返回值含义均在注释上。
- 请注意，`compile`函数现在起需要处理`flags`参数了。

评测器的使用方法

请参见第一次实验文档。
此外，输入`./judge-xxx --help`可以查看程序内置的使用帮助。