概念定义

评测程序：用于编译用户代码、评测用户代码是否正确、判断时间内存是否超限（可选）、将结果输出的程序

评测机：用来调度评测程序、同步依赖文件和用户代码、处理错误、返回结果的程序，是评测程序的父进程。评测机与成绩系统进行对接。评测程序在评测时无需考虑评测机的存在（即评测机对评测程序透明）。

评测过程：每个用户每次提交，将会生成一个评测过程。评测过程是对「用户提交代码」「分发评测任务」「同步评测文件」「运行评测程序」「评测测试点是否运行正确」「反馈评测结果并记录」等一系列工作的抽象概括。各个模块分别负责评测过程中的各个部分，比如成绩系统用来处理用户提交的代码并分发评测任务，评测机负责同步文件和运行评测程序，评测程序负责对评测点进行评测打分等。

评测进程：用于运行评测程序的进程，是评测机创建出的子进程，进程的工作目录中有待评测的代码。

评测点：评测打分的最小单位，实际可能对应多组测试样例。有通过、未通过、未评测三种状态。

依赖：评测程序运行时所要动态加载的依赖库，将会被评测机自动复制至评测程序的工作目录中。原则上这部分代码是**与具体实验无关、与具体测试点无关**的，也不是应该被频繁更新的部分。

用户代码：学生所编写的实验代码+内核框架等，也就是用户所上传提交的文件。

成绩系统接口（API）：评测程序用于向成绩系统发送评测结果的 API，调用时无需考虑评测机的存在（评测机对评测程序透明），而更像直接向成绩系统发起的调用。

API

```
def begin_compile():

def compile_error(reason=""):

def end_compile(warning="No warning"):

def begin_judge():

def judge_error(reason=""):
```

```
def pending_testpoint(test_point_id):

def pass_testpoint(test_point_id):

def reject_testpoint(test_point_id,reason):

def end_judge():

# 这个函数是用来辅助评测程序调试用的，在开始处调用这个函数后，成绩系统 API 将不会将结果输出至标准输出流，以方便评测程序编写者打印调试信息。
def toggle_output_to_file():
```

评测程序将所处位置

（相对与学生代码根目录）

流程

在评测程序运行前，评测所需的静态依赖（如自身的依赖、成绩系统接口等）、用户代码、评测程序自身，都将会被装在到一个临时文件夹中（详情请看上一章的配图）。在评测程序运行时，工作目录将会被设为这个临时目录。

评测程序在编写时无需考虑成绩系统的存在，也无需考虑自己是在评测哪个学生的程序，只需在恰当时候调用成绩系统的 API 函数以给予成绩系统反馈即可。在评测程序调用 end_judge()前，成绩系统不会对评测程序的运行进行任何干涉。

如果评测测试点时出现了与**当前测试点有关**的运行时错误（RuntimeError），则调用 reject_testpoint(id,reason)，在 reason 中以字符串的形式向用户反馈错误信息（如：RuntimeError:越界访问）。**reject_testpoint 函数不会终止当前评测进程。**

同理，内存超限、超时等**需让该评测点被判为「错」**的情况，也应该调用 reject_testpoint(id,reason)来标识这个测试点是错的，错误的原因（比如 TLE，MLE）可以通过 reason 参数传回来。

如果评测时出现**评测程序自身无法 handle** 的错误，比如评测机内存不够、评测程序读写文件时权限不够、自身的 BUG 导致出了异常等情况，那么可以给出 judge_error(reason)，此时成绩系统将会终止本次评测。**judge_error 函数会将当前评测过程设为终止（但是不会杀掉评测程序的进程，你仍可进行清理工作，只是不再接受该评测进程的信息了）**

PS:所有*_error()函数都会终止当前的评测过程(即评测机不再接受该评测过程后续发的消息），但是进程不会杀掉，你仍然可以进行清理工作。

其余的可能的错误，都由评测机自行处理。

评测程序示例

示例仅供演示大致流程

```
import os
from .score_system import begin_judge, \
    begin_compile, \
    end_judge, \
    pass_testpoint, \
    pending_testpoint, \
    reject_testpoint, \
    compile_error, \
    end_compile, \
    judge_error

begin_compile()
compile_msg = os.popen("make").read()
if compile_msg=='error':
    compile_error(compile_msg)
    exit()
end_compile(compile_msg.warning)
begin_judge()
qemu_msg = os.popen("make qemu").read()
if qemu_msg=="error":
    judge_error(qemu_msg)
    exit()
test_point_id_list = [1,2,3]
for id in test_point_id_list:
    pending_testpoint(id)
    # 在此进行评测
    if error:
        judge_error(reason=reason)
    if success:
        pass_testpoint(id)
    else:
        reject_testpoint(id,reject_reason)
end_judge()
# 关闭QEMU
```