

Debug Lab 1

SEP 课程组

2024 年 10 月 26 日

1 Intro

在本 Lab 中，你需要修复 5 个有问题的程序中的 BUG，使得程序能正确执行。

5 个程序分别存放在 P1, P2, P3, P4, P5 文件夹下。每个文件夹下会有一个 `px.cpp` (`x` 代表一个数字，如果 `x` 是 1 就是 `p1.cpp`)。请注意你只能修改 `px.cpp` 以修复 BUG。在对代码进行评测时，我们会使用原本的文件加上你的提交进行编译。

为了防止你通过重写代码来解决问题，我们会使用 Unix Shell 的 `diff` 工具来比较你上传的 `px.cpp` 和原本的 `px.cpp`。每一题都有各自的 `diff` 输出行数的限制。这个限制通常留有足够的余量，同时，我们会在评测前使用统一的代码格式化工具（代码格式就在目录下的 `.clang-format`，你可以在写代码的时候就使用！）格式化你的代码。

在最终评测的时候，我们只会用这样统一的标准衡量不同的代码行，空行、数字、以及其他的行都不会被统计。因此，你不必深究代码风格对答案的影响，也不用为只改了几行就修复了 BUG 感到疑惑。

Unix `diff` 的输出如下：

```
> diff demo.cpp demo.2.cpp
3a4,6
> if (n == 0) {
> return 0;
> }
7,8c10
< int main(int argc, char *argv[]) {
<
---
```

如果是 windows 用户，可以使用如下工具进行替代来检查自己做了哪些修改（比如有没有加空格）。不过要注意 `fc.exe` 的输出包含一些空行，所以输出行数通常比 `diff` 多。

```
fc.exe .\p3.cpp .\p3_answer.cpp | Measure-Object -line
```

最后, 每一道题目都有一些 Hint 来帮助你找到调试的思路, 不妨先尝试自己找找 BUG, 走投无路之时再来寻求帮助:)

P1: Easy Link

问题描述

助教最近在研究 Qlink 的实现。为了做些练习, 助教写了一个小程序, 想完成如下目标:

给定一个 $n \times m$ 的矩阵, 矩阵由整数 0 或 1 组成, 0 表示可以通行, 1 表示无法通行。同时给定一个起点 $(x_{\text{start}}, y_{\text{start}})$ 和终点 $(x_{\text{end}}, y_{\text{end}})$, 求能否从起点开始, 在只经过可以通行的矩阵格子的情况下, 到达终点 (假设起点和终点都可以通行)。

如果能到达则输出 1, 否则输出 0。

样例

	0	1	2
0	1	1	0
1	0	1	0
2	0	1	0

样例一:

输入:

```
3 3 2 0 0 2
1 1 0
0 1 0
0 1 0
```

输出:

```
0
```

分析：矩阵如上， $(x_{\text{start}}, y_{\text{start}}) = (2, 0)$ ， $(x_{\text{end}}, y_{\text{end}}) = (0, 2)$ 。 $(2, 0)$ 代表左下角的点， $(0, 2)$ 代表右上角的点。显然它们被橙色的无法通行的格子隔开了。

样例二：

输入：

```
3 3 2 2 0 2
1 1 0
0 1 0
0 1 0
```

输出：

```
1
```

解释：矩阵如上， $(x_{\text{start}}, y_{\text{start}}) = (2, 2)$ ， $(x_{\text{end}}, y_{\text{end}}) = (0, 2)$ 。 $(2, 2)$ 代表右下角的点， $(0, 2)$ 代表右上角的点。显然没有东西隔开它们。

你的任务

助教写了一个程序，放在 P1 文件夹下，助教发现这个程序总是挂掉，请你帮助解决这个问题。

Hint：程序挂掉会不会是因为内存访问越界？

经过你修改的 p1.cpp 与原本的 p1.cpp 的 diff 输出应少于 20 行。

P2: 奇怪的二分搜索

问题描述

助教学了一点软院的算法课，决定写点 Leetcode 玩玩，打开了一道简单题准备速通。题目要求写一个正确的二分搜索算法。输入一共 3 行，第一行传入一个数字 n ，表示一共有 n 个数字，第二行传入 n 个从小到大有序数字，第三行传入一个数字，表示需要查找的目标。函数返回一个数字，表示目标数字在数组中的下标，如果没有找到，应该返回 -1。

样例

输入：

```
5
37705 44217 57745 58613 88482
58613
```

输出：

```
3
```

你的任务

助教的程序一开始不能通过测试用例，觉得有点奇怪，针对报错又改了改，可以通过 99% 的测试用例了，但还差几个数据集没法通过，请你帮忙解决这个问题。

Hint：剩下的数据集是有什么情况没考虑到吗？

经过你修改的 `p2.cpp` 与原本的 `p2.cpp` 的 `diff` 输出应少于 **20** 行。

P3: 括号不匹配也能工作的计算器

问题描述

有一天，助教在实验室里闲逛，看到编译课的助教正在为编译原理 Lab 而抓狂，于是心血来潮写了一个简单的计算器，选择了**下推自动机**的设计模式。整个计算器只支持加减乘除、括号以及浮点数，并且只能处理有限层次的括号嵌套。

整个程序会逐步解析用户的输入，并用一个栈来保存输入的数据值和操作符。具体来说，当读到数据时，会往栈中压入一个新的数字，继续读取到操作符时，会把操作符和栈顶的数字绑定。每次读到操作符的时候，都会比较当前操作符的优先级和栈顶元素的操作符优先级，如果当前操作符的优先级没有栈顶操作符的高，就执行栈顶的操作符，得到新的值后再压栈，并把当前的操作符绑定到栈顶。

括号的优先级最高，因此会直接压入栈中。左括号对应的操作符是右括号，这样就能够把最右括号匹配起来。遇到右括号的时候，我们会不断地把栈中的元素出栈，直到找到最近的一个左括号为止。

上述的这个过程会逐步地计算出整个表达式的结果，这样我们就得到了一个计算器。

你的任务

助教的程序工作得很好，输入正确的数据总能得到正确的输出，错误的字符也可以正确地报错。然而，当助教准备随便编一个 BUG 给大家的时候，突然惊奇地发现，在左右括号不匹配的时候，计算器居然也可以正确地给出答案。例如：

```
> (((((1+1)))
= 2.000000
```

请你帮助助教找出这个奇怪的 BUG。

Hint：单步调试看看，括号太多的时候究竟发生了什么？

经过你修改的 `p3.cpp` 与原本的 `p3.cpp` 的 `diff` 输出应少于 20 行。

P4: Git Chaos

问题描述

助教想到评测作业的工作量很大，非常苦恼，花了 2.5 小时写好了自动化评测系统。现在，为了更好地整理评测系统输出的相关信息，助教决定写一个简单的日志过滤程序，来更好地检测评测系统运行的状态。

首先，助教实现了一个能够根据输入检测日志，并且根据需要删除掉过时的日志以及重要程度不高的日志（例如，`INFO` 级别或 `DEBUG` 级别的日志往往打印一些系统日常状态的信息）。这样，就可以把日志文件输入到程序中进行分析，并打印出例如 `ERROR` 之类的重要日志了。

这个系统经过了多次的修改，助教为了能够让整个开发的过程更加顺畅，使用了 `git` 作为版本管理工具，并从 `master` 分支开始根据需要创建了很多新的分支。

首先，基本的功能在 `master` 分支中实现，并有了第一次 `commit`(`init: basic implementation of LogAnalyzer`)。之后，助教希望做一些测试，需要一个自动生成大量日志的工具，于是切换到 `feat/gen` 分支实现了一个简单的日志生成器。此时有了第二个 `commit`(`feat: log generator`)。接着助教就把 `feat/gen` 分支直接合并到了 `master`。`master` 分支后续继续修改。

然而，助教突然发现 `feat/gen` 分支有一些 bug 没有处理好，为了方便，就 `revert` 掉了 `feat/gen` 分支的合并。在两个分支继续开发后，助教觉得已经把 bug 修好了，可以合并到主线并进行一些测试。

你的任务

首先，助教希望你帮忙先把 `feat/gen` 正确地合并到 `master`。你需要考虑如何处理当前项目的情况，因为 `revert` 操作只会帮忙回滚文件的修改状态，但会生成新的 `commit`，并且保留了原来的 `commit` 操作。你可能需要 `the revert of revert` 来帮助你实现正确的合并。**注意：你不允许使用强制修改的方式改变已有的任何一个 `commit`。**

恢复好整个项目之后，你可以开始用 `gen` 程序生成数量比较大的 `log`，然后传递给 `p4` 进行处理。但是助教发现，有时候程序会崩溃。你的第二个任务是找出问题，修复程序中的 BUG。

Hint: 这是 `revert` 的一种特殊情况，Linus 曾经对这样的问题进行过详细的解释：github.com/git/git/blob/master/Documentation/howto/revert-a-faulty-merge.txt

Hint: 由于两次 `merge` 中间还有其他的一些操作，所以 `merge` 的时候还需要正确地处理冲突文件。你需要保证最后的 `.gitignore` 是这样的：

```
.DS_Store
build/
.cache/
.idea/
cmake-build-debug/
cmake-build-release/
```

Hint: `git log --graph --all` 会对你的调试很有帮助

Hint: 先用 `gen` 找一个会导致崩溃的例子，然后再单步调试试试。

Hint: 可以先用 `gen` 生成出测试用例，并保存到文件中。再把文件传递给 `p4` 来测试：

```
~> ./cmake-build-debug/gen > logs.txt # 输出重定向，生成log文件
~> ./cmake-build-debug/p4 < logs.txt # 输入重定向，把生成好的log传入p4
~> ./cmake-build-debug/gen | ./cmake-build-debug/p4 # 管道传输
~> ./cmake-build-debug/gen 2000 | ./cmake-build-debug/p4 # 自定义log的数量
```

如果你更喜欢使用 CLion 这样的 IDE，也可以使用 IDE 自身提供的重定向功能。然而，由于 CLion 只支持输入重定向，你需要手动修改 `gen.cpp` 的输出，使其输出到特定的文件。不用担心对 `gen.cpp` 的修改会影响评测，我们会使用原版的 `gen.cpp` 进行测试。

你可以这样修改 `gen.cpp`，使其 `std::cout` 的输出变为文件输出：

```
// 打开一个文件输出流
std::ofstream file("output.txt");

// 保存原始的 std::cout 缓冲区
std::streambuf *coutbuf = std::cout.rdbuf();
// 将 std::cout 的缓冲区重定向到文件输出流
std::cout.rdbuf(file.rdbuf());

// 输出内容将会写入文件
std::cout << "This line will be written to the file!" << std::endl;

// 恢复原始的 std::cout 缓冲区
std::cout.rdbuf(coutbuf);
// 关闭文件
file.close();
```

CLion 的输入重定向功能可以参考这个 [\[网页\]](#)。

经过你修改的 p4.cpp 与原本的 p4.cpp 的 diff 输出应少于 20 行，git 产生的 log 图像与正确答案的 diff 输出应少于 40 行

P5: C++ 是最好的语言吗

问题描述

去年的助教觉得 C++ 是世界上最好的语言，今年的助教看了直摇头。为了让学生们切身体验，助教决定写一个小程序来体现 C++ 之“美”。

助教的程序想要依次完成三个任务：

1. 创建三个栈，编号为 0, 1, 2
2. 反复向栈中填入数字
3. 反复弹出栈中元素直到栈为空

程序输入如下：

- 一个整数 n ，代表接下来会有 n 行输入
- n 行输入，每一行由两个数字 i 和 x ，代表向编号为 i 的栈填入数字 x

预期输出如下：

- 三行输出，每行为对编号为 i 的栈不断进行出栈的结果

样例

输入：

```
5
0 2
0 3
1 4
2 5
1 1
```

输出：

```
stack 0: 3 2
stack 1: 1 4
stack 2: 5
```

解释：

对编号为 0 的栈依次入栈 2 和 3，所以出栈的结果是 3 和 2。其他栈同理。

你的任务

助教写了一个程序，但发现程序总是无法正确输出，请你将程序改对。

Hint: 如果你对 C++ 不是特别熟悉，可以通过打印 log，单步执行等方式来确认 p5.cpp 定义的诸多函数中究竟有哪几个以怎样的顺序被执行了。

Hint: 如果你无法理解为什么程序为这么执行，可以自行查阅关于 C++ 的左值、右值、临时量、生命周期等概念的资料

Hint: 哦！或许应该重新看看上学期那该死的 Lab3 :-(

经过你修改的 p5.cpp 与原本的 p5.cpp 的 diff 输出应少于 20 行。

提交格式

你需要上传一个名为 debug_lab.7z 的压缩包,压缩包解压出来应是一个名为 debug_lab 的文件夹。该文件夹中包含你修改过的 px.cpp 文件。此外，为了检验你是否正确地完成了 git 的操作，你还需要提交完整的 P4 文件夹，文件结构应如下所示：

```
debug_lab/
├── p1.cpp
├── p2.cpp
├── p3.cpp
├── p4.cpp
├── p5.cpp
└── P4/
    └── ... (P4 的所有文件，注意一定要有.git/文件夹)
```

确保你的提交格式严格按照上述要求，以便我们能正确处理和评估你的作业。

注意：请不要提交编译产生的各种额外文件，例如 build、cmake-build-debug 之类的文件夹，其中可能包含了大量与构建相关的缓存文件（有时候高达数百 MB！），导致评测超时。

评测相关信息

五个题目独立评分，彼此之间互不影响。如果你的时间不够充裕，可以只完成一部分任务。但是，**必须保证提交格式完整，缺少文件的情况下无法评测。**

请注意：本次 Lab 不开放多次评测，会在提交时间结束后统一评测。可以多次提交，但以最后一次提交为准打分。