小型代码管理系统的实现方式

灵感来源于多校第五场的K题。同样是大作业一样的题目，与之前不同的是，本次的题目有标程。

题目

与其他人一起编写代码有时意味着浪费时间——至少你会花很多时间来合并不同的分支并测试它们是否存在潜在的bug。有时。git标记的区别只是不同的编码风格！

```
#include <bits/stdc++.h>

int main()
{
<<<<<<< branch1
    printf("Hello world!");
=======
    puts("Hello World!");
>>>>>>> branch2
}
```

下面是一个合并helloworld的例子。branch1上的一个使用printf。但是第二个程序员更喜欢puts。好吧，解决这个问题的一个办法是，在不同的预设开启时，做一些适当的定义，让事情正常运作。最简单的方法之一是将git的标记改为define。也就是说：

```
#include <bits/stdc++.h>

int main()
{
#ifdef branch1
    printf("Hello world!");
#else
    puts("Hello World!");
#endif
}
```

然而，有时git不会标记一个精确的范围，以简单的方式替换标记不会得到一个最短的答案。如果add定义不同，以下代码可能会更短。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int a, b;
<<<<<<< branch1
    cin >> a >> b;
=======
    scanf("%d%d", &a, &b);
```

```
>>>>>>> branch2
    if (a < 0 || b < 0) return 1;
<<<<<<< branch1
    cout << a + b << endl;
=======
    printf("%d\n", a + b);
>>>>>>> branch2
}
```

定义和限制

你的任务是输出一些最短的合并答案，一些C++代码。

- 文件中将有1个或多个冲突。冲突用与示例代码完全相同的格式标记。也就是说：
  - 开头用“«««< branch1”
  - “ ======= ”用于分隔冲突。
  - “»»»> branch2”表示结束。
- 标记之间有0行或更多行代码。
- 代码中不会出现“«««< branch2”或“ »»»> branch1”
- 代码中不会出现嵌套冲突。
- 没有其他行以“ « « « < ”、“ ======= ”或“ » » » > ”开头。
- 输入文件不一定是git的输出，也不能保证标记的精确性：它可能像后面的示例一样包含“ 令人惊讶 ”的输出。但是，它将严格遵循格式。
- 您可以用（仅）以下命令重新排列代码，只要它们符合预处理器标准:

```
#ifdef branch1
#ifdef branch2
#else
#endif
```

- 除了上述指令之外，您不应该插入任何其他指令，也不应该在代码中预处理任何指令。
- 代码中没有其他的#if#ifdef#endif指令，也没有#define#undef与“ branch1”“branch2”相关，因此您不需要处理与普通代码不同的预处理器指令。
- 两个定义“branch1”或“ branch2”将一次打开一个。
- 空行也应算作行，并且处理后的输出应该与两个文件完全匹配。但是，允许输入文件在EOF之前以单个'\n'结尾，并且在'\n'之前没有额外的空格。
- 输入文件的字符集是所有可见的ASCII字符（从33到126□□'\n'□'\t'和空格（32）。换行符被授予“\n”而不是“\r\n”
- “ 最短 ”表示代码行数最少。不一定要输出以字节为单位的最短答案。

输入输出描述

输入文件最多包含4000行代码或git标记，每行最多256个字节，包括“\n”

当然，至少有一个冲突和至少一个代码行。

输出具有最少行数的代码，当不同的定义on时，将与两个分支完全匹配。如果有多个答案，请打印任何答案。

检查器将尝试修复一些令人不快的问题，包括忽略行尾的空格和文件末尾的额外换行符。不过，建议您将答案以样本格式输出，以避免由于呈现错误而导致的错误答案。

输出必须少于5000行，每行最多300字节，包括'\n'，否则，您将收到错误的答案判决。

样例一：

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
<<<<<<< branch1
    cin >> a >> b;
=======
    scanf("%d%d", &a, &b);
>>>>>>> branch2
    if (a < 0 || b < 0) return 1;
<<<<<<< branch1
    cout << a + b << endl;
=======
    printf("%d\n", a + b);a
>>>>>>> branch2
}
```

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
#ifdef branch1
    cin >> a >> b;
    if (a < 0 || b < 0) return 1;
    cout << a + b << endl;
#else
    scanf("%d%d", &a, &b);
    if (a < 0 || b < 0) return 1;
    printf("%d\n", a + b);a
#endif
}
```

样例二：

```
<<<<<<< branch1
int main() {
    return 0;
}
=======
int main() {
}
>>>>>>> branch2
```

```
int main() {
#ifdef branch1
    return 0;
#endif
}
```

样例三：

```
<<<<<<< branch1
int main() {}
=======
int main() {}
>>>>>>> branch2
```

```
int main() {}
```

备注

以输入文件2为例，让我们来验证输入文件的格式是否正确：

“«««< branch1\nint main() {\n return 0;\n}\n=======\nint main() {\n}\n»»»> branch2\n”

一个可能的输出（作为样本输出2）是：

“int main() {\n#ifdef branch1\n return 0;\n#endif\n}\n”

请注意，如果在Windows上运行测试，则在使用getline□□时必须注意到“\r”问题。测试数据将不包含任何'\r'□