

# 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

题意：合并一些代码，使代码长度最短，输出方案。

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#ifdef/ifndef/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”

## 做法

大模拟。

先预处理出两个文件，然后 dp[i][j][0/1/2] 表示第一个文件第 i 行，第二个文件第 j 行，在公用/ifdef 1/ifdef 2 内，切换有代价 1[]dp n m 0 是答案[]dp 可以存在 short 里。

## 通关代码

只使用了C语言。

```
#include<stdio.h>  
#include<string.h>  
  
const char op1[]="<<<<<< branch1";
```

```
const char op2[]="=====";  
const char op3[]=">>>>>> branch2";  
  
struct pp  
{  
    int first;  
    int second;  
};  
  
struct pp makepp(int a,int b)  
{  
    struct pp temp;  
    temp.first=a;  
    temp.second=b;  
    return temp;  
}  
  
struct pp op1hash,op2hash,op3hash;  
  
char s1[4007][267],s2[4007][267];  
int len1[4007],len2[4007];  
char s[267];  
int n1=0,n2=0;  
  
struct pp hash1[4007],hash2[4007];  
  
short f[4007][4007][3],g[4007][4007][3];  
  
struct qq  
{  
    struct pp first;  
    int second;  
};  
  
struct qq makeqq(struct pp a,int b)  
{  
    struct qq temp;  
    temp.first=a;  
    temp.second=b;  
    return temp;  
}  
  
struct qq que[4007];  
  
int cnt=0;  
  
int main()  
{  
    op1hash=makepp(0,0);  
    int i;  
    for(i=0;i<strlen(op1);i++)
```

```
{
    op1hash.first=(1LL*307*op1hash.first+op1[i]+1)%998244353;
    op1hash.second=(1LL*307*op1hash.second+op1[i]+1)%100000007;
}
op2hash=makepp(0,0);
for(i=0;i<strlen(op2);i++)
{
    op2hash.first=(1LL*307*op2hash.first+op2[i]+1)%998244353;
    op2hash.second=(1LL*307*op2hash.second+op2[i]+1)%100000007;
}
op3hash=makepp(0,0);
for(i=0;i<strlen(op3);i++)
{
    op3hash.first=(1LL*307*op3hash.first+op3[i]+1)%998244353;
    op3hash.second=(1LL*307*op3hash.second+op3[i]+1)%100000007;
}
int curstatus=0;
char ch=getchar();
while(ch!=EOF)
{
    int len=0;
    while(ch!='\n'&&ch!=EOF)
    {
        s[len++]=ch;
        ch=getchar();
    }
    struct pp h=makepp(0,0);
    for(i=0;i<len;i++)
    {
        h.first=(1LL*307*h.first+s[i]+1)%998244353;
        h.second=(1LL*307*h.second+s[i]+1)%100000007;
    }
    if(h.first==op1hash.first&&h.second==op1hash.second)
    {
        curstatus=1;
    }
    else if(h.first==op2hash.first&&h.second==op2hash.second)
    {
        curstatus=2;
    }
    else if(h.first==op3hash.first&&h.second==op3hash.second)
    {
        curstatus=0;
    }
    else
    {
        if(curstatus!=1)
        {
            len2[++n2]=len;
            hash2[n2]=h;
            for(i=0;i<len;i++)
```

```
        {
            s2[n2][i]=s[i];
        }
    }
    if(curstatus!=2)
    {
        len1[++n1]=len;
        hash1[n1]=h;
        for(i=0;i<len;i++)
        {
            s1[n1][i]=s[i];
        }
    }
}
if(ch!=EOF)
{
    ch=getchar();
}
}
for(i=0;i<=n1;i++)
{
    int j;
    for(j=0;j<=n2;j++)
    {
        int k;
        for(k=0;k<3;k++)
        {
            f[i][j][k]=10007;
            g[i][j][k]=0;
        }
    }
}
f[0][0][0]=0;
f[0][0][1]=1;
g[0][0][1]=0;
f[0][0][2]=1;
g[0][0][2]=0;
for(i=0;i<=n1;i++)
{
    int j;
    for(j=0;j<=n2;j++)
    {
        if(i==0&&j==0)
        {
            continue;
        }
        if(i>0)
        {
            if(f[i][j][1]>f[i-1][j][1]+1)
            {
                f[i][j][1]=f[i-1][j][1]+1;
            }
        }
    }
}
```

```
        g[i][j][1]=-1;
    }
}
if(j>0)
{
    if(f[i][j][2]>f[i][j-1][2]+1)
    {
        f[i][j][2]=f[i][j-1][2]+1;
        g[i][j][2]=-1;
    }
}
if(i>0&&j>0&&hash1[i].first==hash2[j].first&&hash1[i].second==hash2[j].second)
{
    if(f[i][j][0]>f[i-1][j-1][0]+1)
    {
        f[i][j][0]=f[i-1][j-1][0]+1;
        g[i][j][0]=-1;
    }
}
int k1;
for(k1=0;k1<3;k1++)
{
    int k2;
    for(k2=0;k2<3;k2++)
    {
        if(f[i][j][k1]>f[i][j][k2]+1)
        {
            f[i][j][k1]=f[i][j][k2]+1;
            g[i][j][k1]=k2;
        }
    }
}
}
}
que[++cnt]=makeqq(makepp(n1,n2),0);
while(que[cnt].first.first!=0||que[cnt].first.second!=0||que[cnt].second!=0)
{
    int x=que[cnt].first.first;
    int y=que[cnt].first.second;
    int k=que[cnt].second;
    if(g[x][y][k]!=-1)
    {
        k=g[x][y][k];
    }
    else if(k==0)
    {
        --x,--y;
    }
    else if(k==1)

```

```
{
    --x;
}
else
{
    --y;
}
que[++cnt]=makeqq(makepp(x,y),k);
}
for(i=cnt-1;i;i--)
{
    int x=que[i].first.first;
    int y=que[i].first.second;
    int k=que[i].second;
    if(g[x][y][k]!=-1)
    {
        if(g[x][y][k]==0)
        {
            if(k==1)
            {
                printf("#ifdef branch1\n");
            }
            else if(k==2)
            {
                printf("#ifdef branch2\n");
            }
        }
        else
        {
            if(k==0)
            {
                printf("#endif\n");
            }
            else
            {
                printf("#else\n");
            }
        }
    }
}
else
{
    if(k==1)
    {
        int ii;
        for(ii=0;ii<len1[x];ii++)
        {
            printf("%c",s1[x][ii]);
        }
        printf("\n");
    }
    else
```

```
{  
    int ii;  
    for(ii=0;ii<len2[y];ii++)  
    {  
        printf("%c",s2[y][ii]);  
    }  
    printf("\n");  
}  
}  
}  
return 0;  
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team  
Permanent link: <https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:namespace:%E5%B0%8F%E5%9E%8B%E4%BB%A3%E7%A0%81%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F%E7%9A%84%E5%AE%9E%E7%8E%B0%E6%96%B9%E5%BC%8F&rev=1595942320>  
Last update: 2020/07/28 21:18