Lab2 Preprocessor

一、说明

该实验要求实现一个源代码预处理器(preprocessor)来实现代码文件中对宏定义指令(macro directives)的预编译。代码在进行编译之前，预处理器会将代码中的宏定义进行处理。常见的指令有：#include, #define, #undef, #ifdef, #ifndef, #if, #endif 等等。

你可以使用 g++ -E test1.cpp > output.cpp 来在 output.cpp 中查看 test1.cpp 在进行预编译之后的代码。

在这个 lab 中，你需要实现一个预编译处理器（不需要处理 C++标准库的 include）来处理简单场景下的指令预编译。

二、要求

1、 实现：在 lab 中你需要将你的代码写在 lab2.cpp 中，lab2.cpp 中提供了一个入口函数，你需要在这个入口位置实现你的代码。

2、 测试：在 test 文件夹中有两个 cpp 测试文件。test1.cpp 很简单，可以帮助你在早期阶段 debug。test2.cpp 相对比较复杂，需要你仔细辨别各种宏指令处理的场景。

3、 运行：首先你需要编译 lab2.cpp（使用 c++11 标准编译），并且运行编译后的文件。运行之后，会在 test 文件夹中生成两个 test.out.cpp 文件。然后在 test 文件夹中运行 run_tests.sh 文件，在运行这个文件之前确保你在 test 目录中。

三、评分标准

评分项	分值	说明
指令#include	10	不需要处理#include “iostream”
指令#define	10	包括 check1-check5
指令#undef	10	
指令#ifdef	10	
指令#else	10	
指令#ifndef	10	
指令#if	10	
指令#define function(PART 2)	10	
指令#define function(PART 3)	10	对#和##的处理各 5%，对#和##的处理只要能通过测试文件即可
不存在内存泄露问题	10	要求代码不存在内存泄露的问题

四、提交

将写好的 lab2.cpp、test 文件夹以及你的运行截图打包命名为【学号+姓名.**zip**】上传到 ftp。

全局变量：
一个 map 的对象，<map>库中，类似于 Java 的 map
一个 stack 的对象，<stack>库中，类似与 Java 的栈，初始值中有一个 1
Int write，表示可写或不可写，只有 0 和 1，表示 false 和 true

一些主要的方法：
预处理分为三类：
1. include
2. define 和 undef：变量、函数、#、##
3. ifxxx，else， endif

include：
将#include 指令替换为后面文件的内容

Define：
使用一个 map（全局变量，<map>库中）存储标识符和其他内容
对于一般的变量：如#define true 1 key=true，value=false
对于一般的函数（即不包括#和##），因为宏定义中不支持函数的重载，所以有以下两种存储方式，如#define func（argc） 2*argc，①key=func，value=（argc） 2*argc；②key=func（argc） value=2*argc
③对于#和##存储方法和一般函数相同

Undef：
将后面的标识符从 map 中删除，如果上面对于函数的存储，key 值中只存储了标识符的话，那么删除为比较容易，但是存储比较麻烦；但是如果存储的 key 中包括参数列表的话，那么删除比较麻烦

IfXX：
Ifdef：检测 map 中是否存了后面表示的标识符
Ifndef：与 ifdef 相反
If …：将后面的内容转化为判断的条件，具体判断
此处有一个特殊的处理，要用到全局变量中的 stack 和 write：
考虑到以下情况
#if false
#if true
……
#else
……
#endif
……
#else
#if false
……
#else

……
#endif
……
#endif
此处在一个外层的 if 语句中嵌套了两个 if 语句（分别在 if 和 else 两个分支）
每次遇到 if，都要在 stack 中 push 一个 0 或 1，表示该 if 分支的 false 和 true
遇到 endif，要将 stack 中最上面的值 pop 出来
Write 的初始值是 stack 的初始值 1，表示当前遇到的代码都是要写入的
之后，每次遇到 if、else、endif，都要更新，规则如下：
遇到一个 if 语句时，先将 if 条件的 true 或 false push 进 stack，之后，write = stack[len-1] && stack[len-2]
遇到 else 语句时，write = !stack[len-1] && stack[len-2]
遇到 endif 语句，首先从 stack 中将最上层的值 pop 出来，之后 write=stack[len-1]

其他方法：
将源代码中的宏定义替换的一个方法：
具体实现：首先用 string 的 find 方法或者正则表达式匹配的方式，查找该行源代码中是否存在宏定义的标识符，其次要判断是否可以替换，如果存在一下情况是不可以的：
① #define false 0
#define notfalse 1
cout << notfalse << endl;
此处的 notfalse 中的 false 不能替换为 0，然后打印 not0
② #define false 0
cout << “false” << endl;
打印的字符串中的 false 不能替换

整体逻辑：
1.按照传入的 code，一行一行（通过查找\n 的位置，每次取出一行源代码）的处理：
2.按照“\n”取出的一行源代码可能有一下两种情况：
①宏定义中转行，类似：
#define str “hello，\
world”
因为 C++中宏定义是不允许转行的，所以，如果表示转行，会在行的末尾有一个反斜杠，此处可处理，可不处理（如果仅仅要求通过测试的话，忽略这种情况也 OK 的）
②本身代码中间有“\n”，如：cout << “hello,\n,world” << endl;这句话的输出结果为：
hello,
world
所以，需要检测“\n”后面是否有内容，如果有，还需要把后面的内容拼接到前面
3.取出一行源代码之后，要检查是不是宏定义的代码，可以用正则表达式，具体请去网上搜 C++ 正则表达式的使用，也可以用 string 中的 find 来查找，详见 C++教材第二卷的字符串相关的内容（此处，可能会有一下情况需要考虑：①#之前有空格，②# 和后面的 define 等之间有空格，此处的空格可能不止一个，可能有多个，正则表达式匹配可能会相对简洁）
4.如果是宏定义的话，那么分别调用对应的处理方法，其中比较特殊的是，define 方法，

需要在对源代码处理前，将其中的宏定义替换掉之后再存入 map
5．如果不是宏定义，那么首先查看是否需要写入，然后替换其中的宏定义，写入

注意：最终的 out 文件中不应该存在任何宏定义的语句