Preprocesor

Twoim zadaniem jest napisanie bardzo uproszczonej wersji preprocesora języka C++. Twój program powinien realizować dwie funkcje (w podanej kolejności):

1. Każdy komentarz zastępować znakiem przejścia do nowej linii.
2. Odwołania do makr zastępować kodem podanym w ich definicji.

Wszystkie makra jakie powinien obsługiwać Twój program zawierają się w jednej linii kodu i mogą przybierać jedną z dwóch form, bezparametrową albo z parametrami.

Definicja makra bezparametrowego ma następującą postać:

```
#define nazwa zastępujący-tekst
```

Linia ta określa, że każde wystąpienie słowa "nazwa" w kodzie programu powinno zostać zastąpione przez "zastępujący-tekst". Należy tutaj zaznaczyć, że zastępujący tekst nie musi być pojedynczym wyrazem. Twój program podczas zamiany nie powinien wstawiać białych znaków (spacji, tabulatorów) znajdujących się na końcu linii, w której zdefiniowane jest makro oraz białych znaków pomiędzy nazwą makra, a początkiem zastępującego tekstu. Przykład (niech znak podkreślenia oznacza spację):

```
__#define_PIKWADRAT_____(_3.14__*__3.14_)___
```

Każde wystąpienie słowa PIKWADRAT w kodzie powinno zostać zastąpione przez:

```
(_3.14__*__3.14_)
```

Ponieważ nasz preprocesor jest dosyć specyficzny, w tekście zastępującym nigdy nie występują stałe łańcuchów znakowych. Przykład:

```
#define IMIE "Maciej" // Niedozwolone makro
```

Każda linia z definicją makra (czyli ta zawierająca #define) powinna zostać usunięta. Jednocześnie gwarantujemy, że w każdej linii programu wystąpi co najwyżej jedno odwołanie do makra.

W makrach z parametrami zastosowanie mają te same reguły co w makrze bezparametrowym. Jak sama nazwa wskazuje główną różnicą jest możliwość przekazywania do makra argumentów. Składnia makra z parametrami jest następująca:

```
#define nazwa(parametr1, parametr2, ..., parametr10) tekst-zastępujący
```

Makro tego typu może zawierać od 1 do 10 argumentów. Nazwa każdego z parametrów zawsze będzie składała się z pojedynczej litery alfabetu angielskiego. Każde wystąpienie nazwy argumentu w tekście zastępującym powinno zostać zamienione na wartość parametru przekazaną do makra. Przykład (znak podkreślenia oznacza spację):

```
#define_MAX(_a_,__b_)___((a)_>_(b)_?_(a)_:_(b))__
int_x_=_100,_y_=_200;
int_wieksza_=_MAX(___x_+__100,_y__+_2000___);
```

Powyższy kod powinien zostać zamieniony na:

```
int_x_=_100,_y_=_200;
int_wieksza_=_((x_+__100)_>_(y__+_2000)_?_(x_+__100)_:_(y__+_2000));
```

Jak widać początkowe i końcowe białe znaki usuwane są również z wartości parametrów przekazywanych do makra. Dla ułatwienia zakładamy, że wartości argumentów makr nie będą zawierały nawiasów ani stałych łańcuchów znakowych. Przykłady:

```
int wieksz = MAX( (x + 100) , (y + 200) ); // Niedozwolone dodatkowe nawiasy.
#define ECHO(x) puts(x)
ECHO("Test"); // Stała łańcucha znakowego również nigdy nie wystąpi jako parametr.
```

W kodzie mogą występować dwa rodzaje komentarzy znanych z C++:

- Komentarz wieloliniowy /* ... */ - wszystkie znaki pomiędzy /* a */ (włącznie z nimi) zamieniamy na znak przejścia do nowej linii.
- Komentarz jednolinijkowy // ... - wszystkie znaki pomiędzy // a znakiem przejścia do nowej linii (włącznie z nimi) zamieniamy na znak przejścia do nowej linii. Komentarz ten może również występować w formie wielolinijkowej, jeżeli ostatnim znakiem przed znakiem przejścia do nowej linii jest \ W takim wypadku zawartość wszystkich linii, począwszy od znaków // w pierwszej z nich, zamieniana jest na pojedynczy znak przejścia do nowej linii.

Gwarantujemy, że wszystkie kody, które pojawią się na wejściu są poprawnymi kodami C++, które na kompilatorze g++ kompilują się bez błędów i ostrzeżeń (wyjątkiem jest ostrzeżnie dotyczące wieloliniowego komentarza //).

Wejście

Na wejściu znajduje się kod programu w języku C++, który należy przetworzyć zgodnie z zasadami określonymi w treści zadania. Jego długość nie przekracza 1000 znaków.

Wyjście

Na wyjściu należy wypisać przetworzony kod. Uwaga, program sprawdzający przy porównaniach uwzględnia również białe znaki (spacje, tabulatory), proszę zatem nie wykonywać żadnych dodatkowych zmian w przetwarzanym kodzie poza tymi opisanymi powyżej.

Przykład 1

Wejście (pobierz):

```
#include <cstdio>

#define MAX(a, b) ((a) > (b) ? (a) : (b)) /*
 Zwraca wieksza liczbe
 */
#define MIN( a , b )  ( ( a ) < (  b  ) ?  (a ) : (  b  )) //Mniejsza

#define MWWSIP 5

int main() {
 int a = 1, b = 2, c = 3, d = 4;
 int w = MAX(   a + c   ,   d - b );
 int w2 = MIN(a,d);
 printf("%d %d\n", w, w2);
```

```
 printf("To juz %d Mistrzostwa WWSI w Programowaniu!\n", MWWSIP);
 return 0;
}
```

Wyjście (pobierz):

```
#include <cstdio>




int main() {
 int a = 1, b = 2, c = 3, d = 4;
 int w = ((a + c) > (d - b) ? (a + c) : (d - b));
 int w2 = ( ( a ) < (  d  ) ?  (a ) : (  d  ));
 printf("%d %d\n", w, w2);
 printf("To juz %d Mistrzostwa WWSI w Programowaniu!\n", 5);
 return 0;
}
```

Przykład 2

Wejście (pobierz):

```
#include <cstdio>

#define SUM(a,b) ( ( a ) + ( b ) )

int main() {
 const int MYSUM = SUM(2009,2013);
 printf("      //\\\\\n");
 printf("    //  \\\\\n");
 printf("   //    \\\\\n");
 printf("  //      \\\\\n");
 printf(" //        \\\\\n");
 printf("// SUM=%d   \\\\\n", MYSUM);
 printf("==============\n");
 return 0;
}
```

Wyjście (pobierz):

```
#include <cstdio>


int main() {
 const int MYSUM = ( ( 2009 ) + ( 2013 ) );
 printf("      //\\\\\n");
 printf("    //  \\\\\n");
 printf("   //    \\\\\n");
 printf("  //      \\\\\n");
 printf(" //        \\\\\n");
 printf("// SUM=%d   \\\\\n", MYSUM);
 printf("==============\n");
 return 0;
}
```