Problem E. Ignore all my comments

Small input 17 points	Solve E-small
Large input 0 points	Solve E-large

Problem

Good programmers write fabulous comments. Igor is a programmer and he likes the old C-style comments in `/* ... */` blocks. For him, it would be ideal if he could use this style as a uniform comment format for all programming languages or even documents, for example Python, Haskell or HTML/XML documents.

Making this happen doesn't seem too difficult to Igor. What he will need is a comment pre-processor that removes all the comment blocks in `/*`, followed by comment text, and by another `*/`. Then the processed text can be handed over to the compiler/document renderer to which it belongs—whatever it is.

Igor's pre-processor isn't quite that simple, though. Here are some cool things it does:

- The comments the pre-processor reads can be nested the same way brackets are nested in most programming languages. It's possible to have comments inside comments. For example, the following code block has an outer level of comments that should be removed by the comment pre-processor. The block contains two inner comments.

```
printf("Hello /* a comment /* a comment inside comment */
        inside /* another comment inside comment */
        string */ world");
```

After the pre-process step, it becomes:

```
printf("Hello  world");
```

- Igor recognizes comments can appear anywhere in the text, including inside a string `"/*...*/"`, a constant number `12/*...*/34` or even in a character escape `\/*...*/n`

Or more formally:

```
text:
  text-piece
  text-piece remaining-text
text-piece:
  char-sequence-without-/*
  empty-string
remaining-text:
  comment-block text

comment-block:
  /* comment-content */
comment-content:
  comment-piece
  comment-piece remaining-comment
comment-piece:
  char-sequence-without-/*-or-*/
  empty-string
remaining-comment:
  comment-block comment-content

char:
  letters
  digits
  punctuations
  whitespaces
```

Our pre-processor, given a **`text`**, removes all **`comment-block`** instances as specified.

Notes

- Igor only needs to remove the comment in one pass. He doesn't remove additional comment blocks created as a result of the removal of any comment

block. For example:

```
//*no recursion*/* file header */
```

should generate:

```
/* file header */
```

- The * character in any /* or /*cannot be re-used in another /* or */. For example the following does **NOT** form a proper comment block

```
/*/
```

Input

A text document with comment blocks in /* and */. The input file is valid. It follows the specification of **text** in the problem statement. The input file always terminates with a newline symbol.

Output

We only have one test case for this problem. First we need to output the following line.

```
Case #1:
```

Then, print the document with all comments removed, in the way specified in the problem statements. Don't remove any spaces or empty lines outside comments.

Limits

The input program contains only:

- Letters: a-z, A-Z,
- Digits: 0-9
- Punctuation: ~ ! @ # % ^ & * ( ) - + = : ; " ' < > , . ? | / \ { } [ ] _
- Whitespace characters: space, newline

Small dataset

The small input contains a program of less than 2k bytes.

Large dataset

The large input contains a program of less than 100k bytes.

Sample

Input

```
//*no recursion*/* file header
**********/**********
* Sample input program *
**********/**********
*/
int spawn_workers(int worker_count) {
  /* The block below is supposed to spawn 100 workers.
     But it creates many more.
     Commented until I figure out why.
  for (int i = 0; i < worker_count; ++i) {
    if(!fork()) {
      /* This is the worker. Start working. */
      do_work();
    }
  }
  */
  return 0; /* successfully spawned 100 workers */
}

int main() {
  printf("Hello /*a comment inside string*/ world");
  int worker_count = 0/*octal number*/144;
  if (spawn_workers(worker_count) != 0) {
    exit(-1);
  }

  return 0;
}
```

Output

```
Case #1:
/* file header
**********************
*/
int spawn_workers(int worker_count) {

  return 0;
}

int main() {
  printf("Hello  world");
  int worker_count = 0144;
  if (spawn_workers(worker_count) != 0) {
    exit(-1);
  }
  return 0;
}
```