#include <iostream>

#include<string>

#include<vector>

#include<cmath>

using namespace std;

const char number = '8';

const char quit = 'q';

const char print = ';';

const string prompt = "> ";

const string result = "= ";

void error();

void calculate();

class Token {

public:

char kind;

double value;

Token(char ch)

:kind(ch), value(0) { }

Token(char ch, double val)

:kind(ch), value(val) { }

};

class Token\_stream {

public:

Token\_stream();

Token get();

void putback(Token t);

void ignore(char c);

private:

bool full;

Token buffer;

};

Token\_stream::Token\_stream()

:full(false), buffer(0)

{

}

void Token\_stream::putback(Token t)

{

if (full)

error();

buffer = t;

full = true;

}

Token Token\_stream::get()

{

if (full)

{

full = false;

return buffer;

}

char ch;

cin >> ch;

switch (ch)

{

case quit:

case print:

case '(':

case ')':

case '+':

case '-':

case '\*':

case '/':

case '%':

case '!':

return Token(ch);

case '.':

case '0': case '1': case '2': case '3': case '4':

case '5': case '6': case '7': case '8': case '9':

{

cin.putback(ch);

double val;

cin >> val;

return Token(number, val);

}

default:

error();

}

}

Token\_stream ts;

double expression();

int factorial(int n,int a)

{

if (n < 0)error();

if (n == 0)return 1;

if (n == 1)return a;

if (n > 0)return factorial(n - 1, n\*a);

}

double primary()

{

Token t = ts.get();

switch (t.kind)

{

case number:

{

Token t\_more = ts.get();

if (t\_more.kind == '!') {

int factrial\_number = int(t.value);

if(factrial\_number==t.value)

return factorial(t.value, t.value);

else error();

}

else {

ts.putback(t\_more);

return t.value;

}

}

case'-':

return-primary();

case'+':

return primary();

case'(':

{

double d = expression();

t = ts.get();

if (t.kind != ')') {

error();

}

else {

Token t\_more = ts.get();

if (t\_more.kind == '!') {

int factrial\_number = int(d);

if (factrial\_number == d)

return factorial(d, d);

else error();

}

else {

ts.putback(t\_more);

return d;

}

break;

}

}

default: error();

}

}

double term()

{

double left = primary();

Token t = ts.get();

while (true)

{

switch (t.kind)

{

case'\*':

left \*= primary();

t = ts.get();

break;

case'/': {

double divisor = primary();

if (divisor == 0)error();

left /= divisor;

t = ts.get();

break;

}

case'%':

{

int i1 = int(left);

if (i1 != left)error();

double divisor = primary();

int i2 = int(divisor);

if (i2 != divisor || i2 == 0) error();

left = i1%i2;

t = ts.get();

break;

}

default:

ts.putback(t);

return left;

}

}

}

double expression()

{

double left = term();

Token t = ts.get();

while (true) {

switch (t.kind) {

case'+':

left += term();

t = ts.get();

break;

case'-':

left -= term();

t = ts.get();

break;

default:

ts.putback(t);

return left;

}

}

}

void Token\_stream::ignore(char c) {

if (full&&c == buffer.kind) {

full = false;

return;

}

full = false;

char ch = 0;

while (cin >> ch)

if (ch == c) return;

}

void clean\_up\_mess()

{

ts.ignore(print);

calculate();

}

void error() {

cout << "error" << endl;

clean\_up\_mess();

}

void calculate()

{

while (cin)

{

cout << prompt;

Token t = ts.get();

if (t.kind == print)

t = ts.get();

if (t.kind == quit)

exit(0);

ts.putback(t);

cout << result << expression() << endl;

}

}

int main()

{

calculate();

return 0;

}