#include<iostream>  
#include<sstream>  
using namespace std;  
  
int order(char op);  
string infixToPofix(string infix);  
int pofixComput(string pofix);  
  
int main()  
{  
 string input;  
   
 while(getline(cin, input))  
 {  
 string pofix = infixToPofix(input);//中序式轉後序式   
 cout << pofixComput(pofix) << endl;//後序式運算   
 }  
}  
  
int order(char op)  
{  
 switch(op)  
 {  
 case'(' :  
 return -1;  
 break;  
 case '+' :  
 case '-' :  
 return 0;  
 break;  
 default :// \* / %  
 return 1;  
 }  
}  
  
string infixToPofix(string infix)  
{  
 istringstream isstream(infix);  
 ostringstream pofix;  
 string word;  
 char opStack[1000];  
 int top\_Op = 0;  
 while(isstream >> word)  
 {   
 if(isdigit(word[0]))  
 {  
 pofix << word << " ";//數字直接輸出   
 }  
 else//運算子   
 {  
 switch(word[0])  
 {  
 case '(' : //左括弧直接進入堆疊   
 opStack[top\_Op++] = word[0];  
 break;  
   
 case ')' : //右括弧 ，輸出至左括弧停止  
 while(opStack[top\_Op-1]!='(')//一個左括弧搭配一個又括弧，top\_Op-1必大於-1   
 {  
 pofix << opStack[--top\_Op] << " ";  
 }   
 top\_Op--;//左括弧不輸出 略過   
 break;  
   
 case '+' : case'-' : case '\*' : case '/' : case '%'://加減乘除餘   
 if(top\_Op-1>=0)//避免超出陣列範圍   
 while(order(opStack[top\_Op-1]) >= order(word[0])&&top\_Op-1>=0)  
 {//持續輸出 堆疊中比自己優先的運算子   
 pofix << opStack[--top\_Op] << " ";  
 //cout << "輸出" << opStack[top\_Op] << "top:" << top\_Op;  
 }  
 //放入堆疊  
 //cout << "放入堆疊";  
 opStack[top\_Op++] = word[0];  
 break;  
   
 default :   
 break;  
 }  
 }  
   
 }  
 //讀完全部 全部輸出直到堆疊為空  
 // cout << "輸出至空" ;   
 while(top\_Op-1>=0)   
 {  
 pofix << opStack[--top\_Op] << " ";  
 //cout << "輸出" << opStack[top\_Op];   
 }  
  
 return pofix.str();   
}  
  
int pofixComput(string pofix)  
{  
 int stack[1000];  
 int top = 0;  
 istringstream read(pofix);  
 string word;  
 while(read >> word)  
 {  
 if(isdigit(word[0]))  
 {  
 stack[top++] = atoi(word.c\_str());  
 }  
 else  
 {  
 switch(word[0])  
 {  
 case '+' :   
 stack[top-2] = stack[top-2] + stack[top-1];  
 top -= 1;  
 break;  
 case '-' :  
 stack[top-2] = stack[top-2] - stack[top-1];  
 top -= 1;   
 break;  
 case '\*' :  
 stack[top-2] = stack[top-2] \* stack[top-1];  
 top -= 1;  
 break;  
 case '/' :   
 stack[top-2] = stack[top-2] / stack[top-1];  
 top -= 1;  
 break;  
 case '%' :   
 stack[top-2] = stack[top-2] % stack[top-1];  
 top -= 1;  
 break;  
 }  
 }  
 }  
 return stack[0];  
}