1. Expression Evaluator

Write an expression evaluator that takes a one-variable expression in the form of a string as input and prints out the value of the expression for different values of the variable.

Assume the variable to be made of one lower case character [a-z]. Following are symbols allowed in the input expression:

Symbol	**Meaning**
+ , - , * , /	Same as in C Programming
^	Power

Data Format

Input Format
First line contains the expression Subsequent lines contain the variable value to be used for evaluation

Output Format
First line contains the expression being evaluated Subsequent lines contain the value the of expression for each corresponding input value

Version 1

The first deliverable will evaluate simple expression. Simple expression expressions do not use parentheses to alter the precedence of operators.

Assumptions

1. There will be only one variable represented by a single character
2. Assume all operators have equal precedence and evaluated left to right
3. Any of the five operators may appear in any order
4. No parentheses
5. Input will come from standard input and output to the screen. No need for file I/O

Example

Input lines
P(x) = x^2 + 4*x – 3
3
4
-1
1000

Output lines
P(x) = x^2 + 4*x – 3
P(3)=36
P(4)=77
P(-1)=-8

File naming conventions

expr_v1_**Rollno**.c (e.g., expr_v1_MT2014001.c) - Roll number should be in upper case

Version 2

The second version is for evaluating an expression in which parenthesis may be used for altering the order of evaluation of arguments.

Assumptions

1. There will be only one variable represented by a single character
2. Except when parenthesis are present, assume all operators have equal precedence and evaluated left to right
3. Parentheses alters the order of evaluation of expressions like in normal programming
4. No nesting of parentheses will be there
5. Any of the five operators may appear in any order
6. Input will come from standard input and output to the screen. No need for file I/O
7. An input line with the number 1000 signals the end of the input. No need to evaluate further.

Example

Input lines
P(x) = (x-2)^3 – (x – 3)
3
4
-1
1000

Output lines
P(x) = (x-2)^3 – (x – 3)
P(3)=1
P(4)=7
P(-1)=-23

File naming conventions

expr_v2_**Rollno**.c (e.g., expr_v2_MT2014001.c) - Roll number should be in upper case

Version 3 - Bonus

The third version is a full-fledged expression evaluator that meets the standard rules of precedence of operators

Assumptions

1. There will be only one variable represented by a single character
2. Assume exponent (^) has the highest operator precedence. Standard rules apply for operator precedence apply for the remaining operators (* / + -)
3. Parentheses alters the order of evaluation of expressions like in normal programming
4. No nesting of parentheses will be there
5. Any of the five operators may appear in any order
6. Input will come from standard input and output to the screen. No need for file I/O
7. An input line with the number 1000 signals the end of the input. No need to evaluate further.

Example

Input lines
P(x) = (x-2*x)^3 – (x – 3)
3
4
-1
1000

Output lines
P(x) = (x-2*x)^3 – (x – 3)
P(3)=3
P(4)=-65
P(-1)=-23

File naming conventions

expr_v3_**Rollno**.c (e.g., expr_v3_MT2014001.c) - Roll number should be in upper case