Exercise

- AOJ0109 Smart Calculator
  English Version
  It is quite same as the case study in this slide.
- AOJ1155 How can I satisfy thee? Let me count the ways...
  English Version

AOJ0109 Smart Calculator

- It is quite same as the case study in this slide.

Code 1/4

```cpp
#include <iostream>
#include <string>
#include <ctype.h>

using namespace std;
typedef string::const_iterator State;

int number(State& s);
int term(State& s);
int factor(State& s);
int expr(State& s);
```

Code 2/4

```
int number(State& s) {
  int ret = 0;
  while(isdigit(*s)) {
    ret *= 10;
    ret += *s - '0';
    s++;
  }
  return ret;
}


int term(State& s) {
  int ret = factor(s);
  while (1) {
    char op = *s;
    if (op == '*') {
      s++;
      ret *= factor(s);
    } else if (op == '/'){
      s++;
      ret /= factor(s);
    } else {
      break;
    }
  }
  return ret;
}
```

Code 3/4

```
int factor(State& s) {
  if (*s == '(') {
    s++;
    int ret = expr(s);
    s++;
    return ret;
  } else {
    return number(s);
  }
}

int expr(State& s){
  int ret = term(s);
  while(1){
    char op = *s;
    if (op == '+') {
      s++;
      ret += term(s);
    } else if (op == '-') {
      s++;
      ret -= term(s);
    } else {
      break;
    }
  }
  return ret;
}
```

Code 4/4

```cpp
int main()
{
  int n; cin >> n;
  while (n--) {
    string str;
    cin >> str;
    str.pop_back();
    State s = str.begin();
    cout << expr(s) << endl;
  }
  return 0;
}
```

AOJ1155 How can I satisfy thee? Let me count the ways...

Remember the step to solve prasing problem:

Step

After making BNF:

1. Write functions named BNF's left value.
    - They always take an argument reference of State.
    - In the case that each mutually call them, you have to write function prototypes.
2. Fill the procedure to parse.

Making BNF

Already given in problem statement:

```
<formula> ::= 0 | 1 | 2 | P | Q | R |
              -<formula> | (<formula>*<formula>) | (<formula>+<formula>)
```

Step1

1. Write functions named BNF's left value.

```
<formula> ::= 0 | 1 | 2 | P | Q | R |
              -<formula> | (<formula>*<formula>) | (<formula>+<formula>)
```

```
int formula(State& s) {
}
```

Step2 1/5

2. Fill the procedure to parse.
   '|' means or, so firstly write if-statement.
   It is good to write to print error for debug.

```
<formula> ::= 0 | 1 | 2 | P | Q | R |
              -<formula> | (<formula>*<formula>) | (<formula>+<formula>)
```

```cpp
int formula(State& s) {
  if (*s == '0' || *s == '1' || *s == '2') {

  } else if (*s == 'P' || *s == 'Q' || *s == 'R') {

  } else if (*s == '-') {

  } else if (*s == '('){

  } else {
    cout << "Error" << endl;
    return -1;
  }
}
```

Step2 2/5

```
<formula> ::= 0 | 1 | 2 ...
```

You only return the value.

Don't forget s++.

```
...
if (*s == '0' || *s == '1' || *s == '2') {
  int ret = *s - '0';
  s++;
  return ret;
}
...
```

Step2 3/5

```
<formula> ::= ... | P | Q | R | ...
```

You only return the value.

The value is saved in vars.

```
int vars[3];
...
else if (*s == 'P' || *s == 'Q' || *s == 'R') {
  int ret = vars[*s - 'P'];
  s++;
  return ret;
}
...
```

Step2 4/5

```
<formula> ::= ... | -<formula> | ...
```

I will make function not_op.

```
...
else if (*s == '-') {
  s++;
  int tmp = formula(s);
  return not_op(tmp);
}
...
```

Step2 5/5

```
<formula> ::= ... | (<formula>+<formula>) | (<formula>*<formula>)
```

- They only have two formulas, so I save them as lval(left value) and rval(right value).
- I will make function and_op and or_op.

```
...
else if (*s == '(') {
  s++;
  int lval = formula(s);
  char op = *s;
  s++;
  int rval = formula(s);
  s++;
  return not_op(tmp);
}
...
```

Supplement

- The functions of not_op, and_op and or_op are made folloing to problem statement.
- Brute-forth of (P,Q,R), so after setting values to P, Q, and R, you eval formula.

Code 1/3

```
#include <iostream>
#include <string>
#include <ctype.h>

using namespace std;
typedef string::const_iterator State;

int vars[3];
int not_op(int x) {
  return 2 - x;
}
int and_op(int x, int y) {
  if (x == 0 || y == 0) return 0;
  else if (x == 2 && y == 2) return 2;
  else return 1;
}
int or_op(int x, int y) {
  if (x == 2 || y == 2) return 2;
  else if (x == 0 && y == 0) return 0;
  else return 1;
}
```

Code 2/3

```cpp
int formula(State& s) {
  if (*s == '0' || *s == '1' || *s == '2') {
    int ret = *s - '0';
    s++;
    return ret;
  } else if (*s == 'P' || *s == 'Q' || *s == 'R') {
    int ret = vars[*s - 'P'];
    s++;
    return ret;
  } else if (*s == '-') {
    s++;
    int ret = not_op(formula(s));
    return ret;
  } else if (*s == '('){
    s++;
    int lval = formula(s);
    char op = *s;
    s++;
    int rval = formula(s);
    s++;
    if (op == '*') return and_op(lval, rval);
    else return or_op(lval, rval);
  } else {
    cout << "Error" << endl;
    return -1;
  }
}
```

Code 3/3

```
int main()
{
  string str;
  while (cin >> str, str != ".") {
    int ans = 0;
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 3; k++) {
          vars[0] = i;
          vars[1] = j;
          vars[2] = k;
          State s = str.begin();
          if (formula(s) == 2) ans++;
        }
      }
    }
    cout << ans << endl;
  }
  return 0;
}
```