OOP Homework Assignment № 2

Frank (Shuxin) Chen, Tsinghua University 08/23/2016

Problem 1

The code for this problem is quite complicated. It took me for almost an hour and a half to finish this problem.

The algorithm I use for this problem is depth-first-search(DFS). In a recursion round, suppose we now have n remaining expressions and values to these expressions. We choose two of them, using one of the for operators +, -, *, / to concatenate the two expressions(if one expression only contains only a number, we do nothing extra, otherwise we should add parentheses to ensure the order of operations). We should note that if the value to the second expression is zero, we cannot use operator / for "division by zero". After the steps above, we can move to the next deeper recursion round.

When we get all the possible solutions, the next thing we should do is delete all the repeated expressions. We can simply use std::sort and std::unique to implement it.

Listing 1: The code for sort and unique.

```
1 std::sort(results.begin(), results.end());
2 auto tail = std::unique(results.begin(), results.end());
3 results.erase(tail, results.end());
```

The result of the sample input is shown as follows. The output order is a little different from the sample output.

Listing 2: The result of sample input.

```
1  Total Methods Count = 20
2  ((10+5)-3)*2
3  ((10-3)+5)*2
4  ((5+10)-3)*2
5  ((5-3)+10)*2
6  (10+(5-3))*2
7  (10+2)*(5-3)
8  (10-(3-5))*2
9  (2+10)*(5-3)
10 (5+(10-3))*2
11 (5-(3-10))*2
12 (5-3)*(10+2)
13 (5-3)*(2+10)
14 2*((10+5)-3)
```

```
15 2*((10-3)+5)
16 2*((5+10)-3)
17 2*((5-3)+10)
18 2*(10+(5-3))
19 2*(10-(3-5))
20 2*(5+(10-3))
21 2*(5-(3-10))
```

In the recursion, we should always use double-type variables to store the expression values, since examples like 1, 5, 5, 5 cannot be solved using only +, -, * and integer /.

Listing 3: The result of another input.

```
1 (5-(1/5))*5
2 5*(5-(1/5))
```

Problem 2

This problem is a classic problem using binary enumeration. We assume that we cannot change the order of the numbers(it is not mentioned in the assignment, but if we can change the order, the time complexity will be higher and the program may not output the result in one day). We define + as 0 and * as 1, and the operator sequence will be an arbitrary binary string with length n-1. Thus, we can enumerate all the possible binary strings with time complexity $O(n)$ if we use depth first search or $O(n \log n)$ if we use decimal numbers from 0 to $2^{n-1} - 1$.

Another important fact we should notice is the result may overflow. In this case, we can substitute + with -, and * with /. For example, when we have no idea about $a + b > max$, we can judge if $a > max - b$ and then calculate $a + b$. Once the results overflows, we break the loop and of course, it will not be a correct solution. Thus, we can use time complexity $O(1)$ to deal with additions and multiplications. The total time complexity is $O(n)$ or $O(n \log n)$. The following shows an example of n = 24 and the time to run the example.

Listing 4: An example in the terminal.

```
1  $ time ./main
2  24 584950345894385
3  9999 9999 9999 9999 9999 9999 9999 9999
4  999 999 999 999 999 999 999 999
5  99 99 99 99 99 99 99 99
6  No
7  593821818611307
8  real    0m0.963s
9  user    0m0.726s
10 sys     0m0.004s
```