HomeWork I

Naifan Gao

Sep. 15, 2017

Test system: Mac os;
Test tools: XCode, Cmake;
Test examples' code are all in ~/01_Naifan_Gao/src;
Answer:

1)

A. In Assignment_01.cpp, I defined two kind of variable:
   double x, float y
   Then, I use cout.precision(i) i = 1,…20 to check the output with different precision.
   As all float and double numbers are stored in binary, so it is actually an infinite binary representation. When we use the precision larger than its real length or its boundary, we'll get a number with more decimal places than the actual one.
B. We set float number with 8 numbers after point, and double with 17 numbers.
   When precision is in the range of [1, 7], the outputs of both float and double are correct as we expected. When precision >= 8, float number goes wrong. It begins to appear with some new decimal numbers. When precision >=17, double goes wrong with the same problem.
   So the correct print precision for float number is 7; correct precision for double number is 16;

2) General rules

If input is positive number: (num1 = num2)

- If either operand type is unsigned long, the other operand type is changed to long;
- If the previous condition not met, and if either operand type is long and the other operand type is unsigned int, the both operand type is changed to unsigned long;
- If the previous two conditions not met, and if either operand type is long, the other operand will be converted to long;
- If the previous three conditions not met, and if either operand type is unsigned int, the other operand will be converted to unsigned int;
- If none of previous conditions not met, both operand type will be converted to int;
- The integer type of the right(num2) will converse to the left type when do the equation;
- If the right side number is larger than the range of the left side, it will be conversed to be the largest number of the left side(num1);

If input is negative number: (num1 = num2)

- Previous six rules of positive input will still work under the condition of negative input;
- The output will be the larger number of type num1 – absolute number of num2 + 1, just convert type num2 to type num1 and calculate num2's complement;
- If the absolute value of num2 out of the range of num1, the output will be zero;

If type num1 is char, then other integer type will be converted as the ASCII. If the value is out the range of 127, it will be represented as \370 in the console.

3) If use streamstringto calculate, but I will write how to calculate it here:
   Input number: 0x3F400000.
   1) Convert it into binary bits: 0011 1111 0100 0000 0000 0000 0000 0000
   2) The first bit is signed bit, so it is a positive float number
   3) Calculate exponent: 0111 1110. It is 126, so the bias value is -1
   4) Calculate fraction: 100 0000 0000 0000 0000 0000, it is 0.5. Consider the leading digit of fraction number is 1, it is 1.5
   5) Use the equation: $f = sign * 2^{exponent} * fraction = 1 * 2^{-1} * 1.5 = 0.75$

4) There will also have a type cast when performs cross-type comparisons, rules are as following:
   - If the either operand type is long double, the other operand type is converted to long double;
   - If the previous condition not met, and either operand type is double, the other operand is converted to double;
   - If the previous two conditions not met, and either operand type is float, the other operand is converted to float;

5) The result is 0 as false. Because INF number is an infinite large number that has out the range of type double, and IND is actually not a real number stored in the memory, represented as NaN. So the comparison of these two number is false, they could not be same.