Ann Henderson

Mr. West

Program Semantics

17 November 2016

Finding the Maximum of Two Numbers Without Using the Comparison Operators

The most common embodiment of the “max” function is the check of the sign of the expression “a – b”. In this case, we cannot use a comparison operator, but we can use multiplication.

Let’s indicate the sign of an expression of “a – b” as “x”. If a – b > = 0, x = 1, otherwise x = 0. Let’s indicate “y” as an inverted value of “x”.

The code will look like:

```
/* inverse 1 в 0 и 0 в 1 */
int inverse(int bit) {
    return 1^bit;
}

/* return 1 if the number is positive or 0 if it is negative*/
int sign(int a) {
    return reverse((a >> 31) & 0x1);
}
```

```
int maxNaive(int a, int b) {
    int x = sign(a - b);
    int y = inverse(x);
    return a * x + b * y;
}
```

This is an almost workable code. The problems start when the stack is overflowed. Suppose that a = INT_MAX - 2 and b = -15. In this case a - b will no longer fit in INT_MAX and it will cause an overflow (value becomes negative).

We can use the same approach, but come up with a different implementation. We need to satisfy the condition x = 1 when a> b. To do this it is necessary to use a more complicated logic.

When is there an overflow a – b? Only when a is a positive number, and b is negative (or vice versa). It is difficult to detect the fact of overflow, but we are able to understand that a and b have different signs. If a and b have different signs, then assign x = sign (a).

Here is how the logic looks like:

1. if a and b have different signs
2. //if a > 0, then b < 0 and x = 1
3. //if a < 0, then b > 0 and x = 0
4. //anyway, x = sign(a)
5. k = sign(a)

6. otherwise

7. k = sign(a - b) // overflow is impossible

The following code implements this algorithm using multiplication instead of comparison operators:

```
int max(int a, int b) {

    int c = a - b;



    int sa = sign(a); // if a >= 0, then 1, otherwise 0

    int sb = sign(b); // if a >= 1, то 1, otherwise 0

    int sc = sign(c); // depends on the overflow a - b



    /* The purpose is to find x, which is equal to 1 if a > b, and 0,
if a < b.

     * if a = b, x doesn't matter */



    // if a and b have equal signs, x = sign(a)

    int use_sign_of_a = sa ^ sb;



    // if a and b have equal signs, x = sign(a - b)
```

```
        int use_sign_of_c = reverse(sa ^ sb);


        int x = use_sign_of_a * sa + use_sign_of_c * sc;
        int y = reverse(x);


        return a * x + b * y;
    }
```

Note that this code is dedicated by the methods and variables for the clarity. This is not the most compact and efficient way of writing this code, but it's clearer in this way.

Thanks for your attention!