Detect if two integers have opposite signs

Given two signed integers, write a function that returns true if the signs of given integers are different, otherwise false. For example, the function should return true -1 and +100, and should return false for -100 and -200. The function should not use any of the arithmetic operators.

Let the given integers be x and y. The sign bit is 1 in negative numbers, and 0 in positive numbers. The XOR of x and y will have the sign bit as 1 iff they have opposite sign. In other words, XOR of x and y will be negative number number iff x and y have opposite signs. The following code use this logic.

```
#include<stdbool.h>
#include<stdio.h>

bool oppositeSigns(int x, int y)
{
    return ((x ^ y) < 0);
}

int main()
{
    int x = 100, y = -100;
    if (oppositeSigns(x, y) == true)
       printf ("Signs are opposite");
    else
      printf ("Signs are not opposite");
    return 0;
}
```

Output:

```
Signs are opposite
```


Source: Detect if two integers have opposite signs

We can also solve this by using two comparison operators. See the following code.

```
bool oppositeSigns(int x, int y)
{
    return (x < 0)? (y >= 0): (y < 0);
}
```

The first method is more efficient. The first method uses a bitwise XOR and a comparison operator. The second method uses two comparison operators and a bitwise XOR operation is more efficient compared to a comparison operation.

We can also use following method. It doesn't use any comparison operator. The method is suggested by Hongliang and improved by gaurav.

```
bool oppositeSigns(int x, int y)
{
    return ((x ^ y) >> 31);
}
```

The function is written only for compilers where size of an integer is 32 bit. The expression basically checks sign of (x^y) using bitwise operator '>>'. As mentioned above, the sign bit for negative numbers is always 1. The sign bit is the leftmost bit in binary representation. So we need to checks whether the 32th bit (or leftmost bit) of x^y is 1 or not. We do it by right shifting the value of x^y by 31, so that the sign bit becomes the least significant bit. If sign bit is 1, then the value of (x^y)>>31 will be 1, otherwise 0.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.

Geometric Algorithms

Popular Posts

Related Tpoics:

- Check if a number is multiple of 9 using bitwise operators
- How to swap two numbers without using a temporary variable?
- Divide and Conquer | Set 4 (Karatsuba algorithm for fast multiplication)
- Find position of the only set bit
- Swap all odd and even bits
- Add two bit strings
- Write your own strcmp that ignores cases
- Binary representation of a given number

Writing code in comment? Please use **ideone.com** and share the link here.

23 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

divyansh8063 • 2 months ago

```
main()
{
int i=-5,j=-7;
```

```
j=j&(1<<31);

if(i==j)
printf("same");
else
printf("diff");
}
```

Reply • Share ›

atiq • 10 months ago

```
// can be done by these ways too..
unsigned int oppositeSign(unsigned int x,unsigned int y)
{
return !(x&y>>31)

//or

return (y&1(<<31))^(x&(1<<31))
}
```

Reply • Share ›

Balasubramanian.N • 2 years ago

In the last line of the last paragraph, it says:

"If sign bit is 1, then the value of (x^y)>>31 will be 1, otherwise 0"

But, if sign extension takes place,wherein the leftmost bits will be filled with 1 i value will be -1 and not 1.

So, I think it is better to say that, if the sign bit is 1, then the value of (x^y)>>31

Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 31 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 2 hours ago

Thanks,
Balasubramanian.N

dexterltd • 2 years ago

why u people simply multiply them.
int of opposite sign when multiple alwyz produce -ve value.
why to complex simple logic.

```
if ( x * y < 0 )
  return true  // means they are opposite signs
return false // they are of same sign
```


dexterltd ➦ dexterltd • 2 years ago

sorry didn't read the question completely.
missed the part "not to use the airthmetic operator.

dexterltd ➦ dexterltd • 2 years ago

I meant why don't you simply multiply . Sorry for the Typo.

Ravi • 2 years ago

Hi,
small addition to Hongliang's idea.

```
 (x^y) >> ((1<<sizeof(int))-1) /* or */
(x^y) >> ~(1<<sizeof(int))
```

makes code independent of compiler used. For the second solution, gcc raise

Reply • Share ›

sudhanshu ➚ Ravi • 10 months ago

@Ravi, The 1<<sizeof(int) doesn&#039t work on my computer atleast

it gives sizeof(int) as 4 and and thus, you get 15 instead of 31 on subtr

Reply • Share ›

Vignesh ➚ sudhanshu • 4 months ago

sizeof() operator returns the size in bytes. so you need to multi
a violation of the prerequisite 'no arithmetic op' :)

Reply • Share ›

GeeksforGeeks • 2 years ago

@Hongliang and @gaurav:

Thanks for your inputs. We have added the new method to the original post.

Reply • Share ›

partik • 2 years ago

Why XOR is more efficient compared to comparison operator?

Reply • Share ›

Sree ram ➚ partik • 2 years ago

XOR is an operator which is done bitwise ...
its more close to hardware because in hardware the numbers are repr
XORing the bits ... and as suggested by kartik

((x ^ y) & 1 << 31); is more efficient ...

Reply • Share ›

Sree ram ➚ partik • 2 years ago

XOR is an operator which is done bitwise ...
its more close to hardware because in hardware the numbers are repr
XORing the bits ... its efficient ...

^ | v • Reply • Share ›

Hongliang • 2 years ago

Nice solution, but it still uses ">" for comparison (you might know that compari
numbers, here it is one without using comparisons:

```
bool opp = ((x ^ y) & >> 31)
```

P.S. actually the webpage for bit hack is worth reading, highly recommended.

^ | v • Reply • Share ›

gaurav ➚ Hongliang • 2 years ago

I think what you are trying to say is
bool opp = (x ^ y) >> 31)

but this would not work as right shift is arithmetic shift in c. So, the resu

((x ^ y) & 1 << 31) would work as suggested by kartik.
[sourcecode language="C"]
/* Paste your code here (You may delete these lines if not writing code

^ | v • Reply • Share ›

Hongliang ➚ Hongliang • 2 years ago

I clicked "Have your say" too fast :-(

The explanation: the sign is determined by the highest bit of the numbe
one can logically XOR it as in the original post, and then check the high

>>31 means to move 31 bit which gives the highest bit. It saves one c

kartik ➚ Hongliang • 2 years ago

@Hongliang: Thanks for suggesting the optimization. I think, the
Following is the complete code.

```
 #include<stdbool.h>
#include<stdio.h>

bool oppositeSigns(int x, int y)
{
    return ((x ^ y) & 1 << 31);
}

int main()
{
    int x = 100, y = -1000;
    if (oppositeSigns(x, y) == true)
       printf ("Signs are opposite");
    else
      printf ("Signs are not opposite");
    return 0;
}
```


Pravesh ➚ kartik • 2 years ago

Kartik, Can you please explain it in deep,

why return ((x ^ y) & 1 << 31 ?

Reply • Share ›

kartik ➚ Pravesh • 2 years ago

@Pravesh: The program is written only for compilers w

Two numbers are negative if the value of their XOR is n
negative numbers is always 1, and the sign bit is the lef
representation.
The expression basically checks whether the 32th bit (c
It does it by doing bitwise and of (x^y) with 1000...0 (1 fo

Reply • Share ›

Abhay • 2 years ago

Just like XOR, multiplication of two numbers would be less than zero if they ha

```
/* Paste your code here (You may delete these lines if not writing c
```

Reply • Share ›

vijay ➚ Abhay • 2 years ago

. The function should not use any of the arithmetic operators.

```
/* Paste your code here (You may delete these lines if not wri
```

Reply • Share ›

Abhay ➚ vijay • 2 years ago

My bad. i missed that part of statement

Reply • Share ›

Dheeraj ➚ vijay • 2 years ago

Also multiplication method is not good because it may cause ar