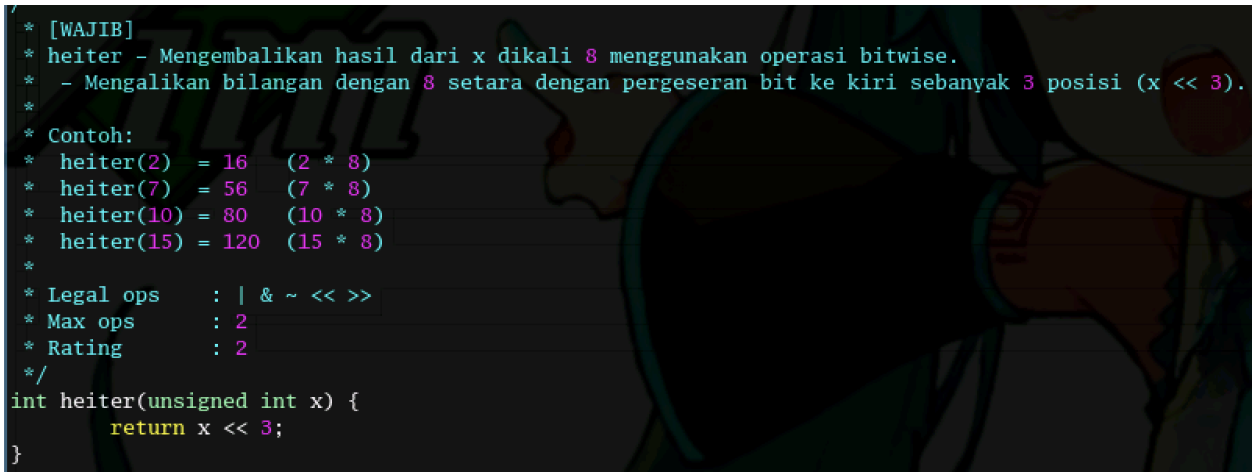
Write Up IF 1230 Praktikum 1
Organisasi dan Arsitektur Komputer

Written by:
strawberry_crepe_cookie (Cikang)
Jingglang Galih Rinenggan 13524095

9/4/2025

1. Heiter



```
 * [WAJIB]
 * heiter - Mengembalikan hasil dari x dikali 8 menggunakan operasi bitwise.
 *  - Mengalikan bilangan dengan 8 setara dengan pergeseran bit ke kiri sebanyak 3 posisi (x << 3).
 *
 * Contoh:
 *  heiter(2)  = 16   (2 * 8)
 *  heiter(7)  = 56   (7 * 8)
 *  heiter(10) = 80   (10 * 8)
 *  heiter(15) = 120  (15 * 8)
 *
 * Legal ops    : | & ~ << >>
 * Max ops      : 2
 * Rating       : 2
 */
int heiter(unsigned int x) {
        return x << 3;
}
```

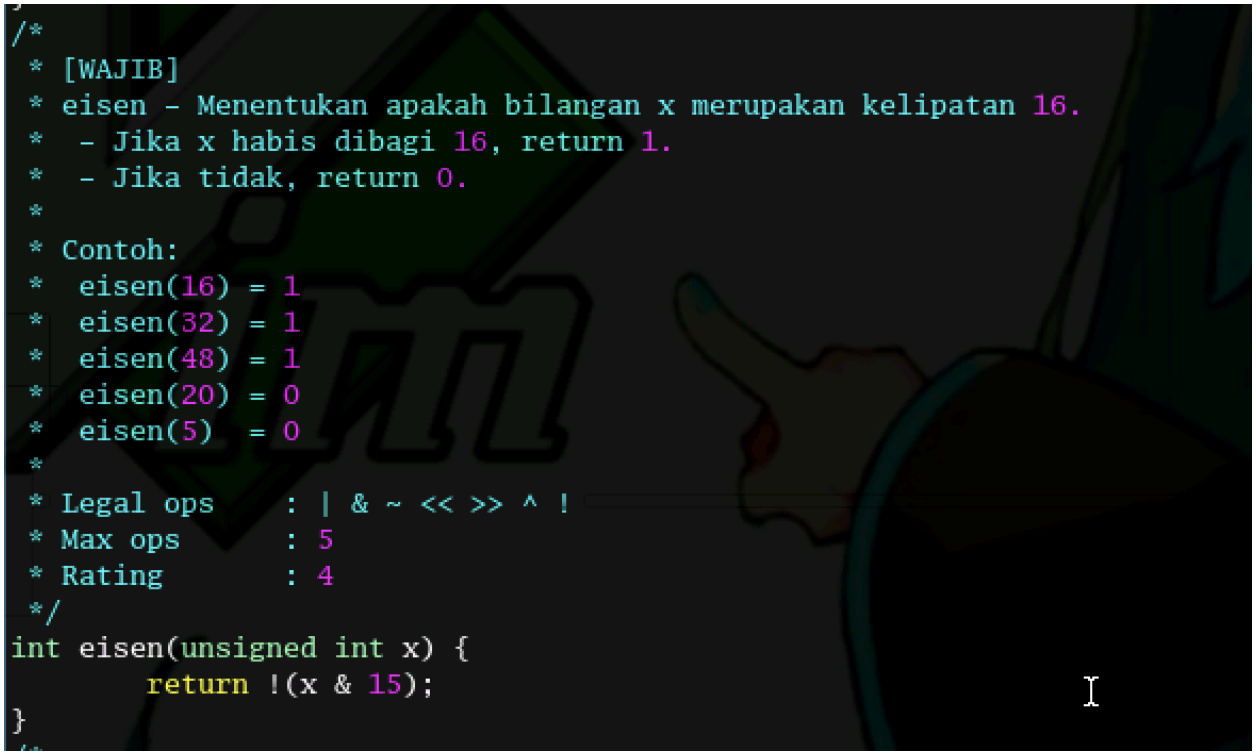
Pada soal ini, diminta untuk mengalikan x dengan 8 tanpa menggunakan operator '*'. 8 merupakan hasil dari 2^x sehingga kita dapat menggunakan operator left shift (<<) yang ekuivalensi aritmatikanya adalah

$x << y == x * 2^x$

Sehingga,

$x << 3 == x * 2^3$

2. Eisen



```
/*
 * [WAJIB]
 * eisen - Menentukan apakah bilangan x merupakan kelipatan 16.
 *  - Jika x habis dibagi 16, return 1.
 *  - Jika tidak, return 0.
 *
 * Contoh:
 *  eisen(16) = 1
 *  eisen(32) = 1
 *  eisen(48) = 1
 *  eisen(20) = 0
 *  eisen(5)  = 0
 *
 * Legal ops    : | & ~ << >> ^ !
 * Max ops      : 5
 * Rating       : 4
 */
int eisen(unsigned int x) {
        return !(x & 15);
}
```

Suatu bilangan yang merupakan kelipatan a ketika a adalah hasil dari 2^x untuk x bilangan bulat, sama saja dengan bilangan yang memiliki bit kosong pada semua [0...(x - 1)]. Kelipatannya yang diminta 16 (2^4) sehingga bit [0..3] harus kosong. Untuk memperoleh bit [0...3] kita dapat mengambil dengan mengoperasikan bit flag dengan yang memenuhi seluruh bit itu, yaitu 15 (0b111).

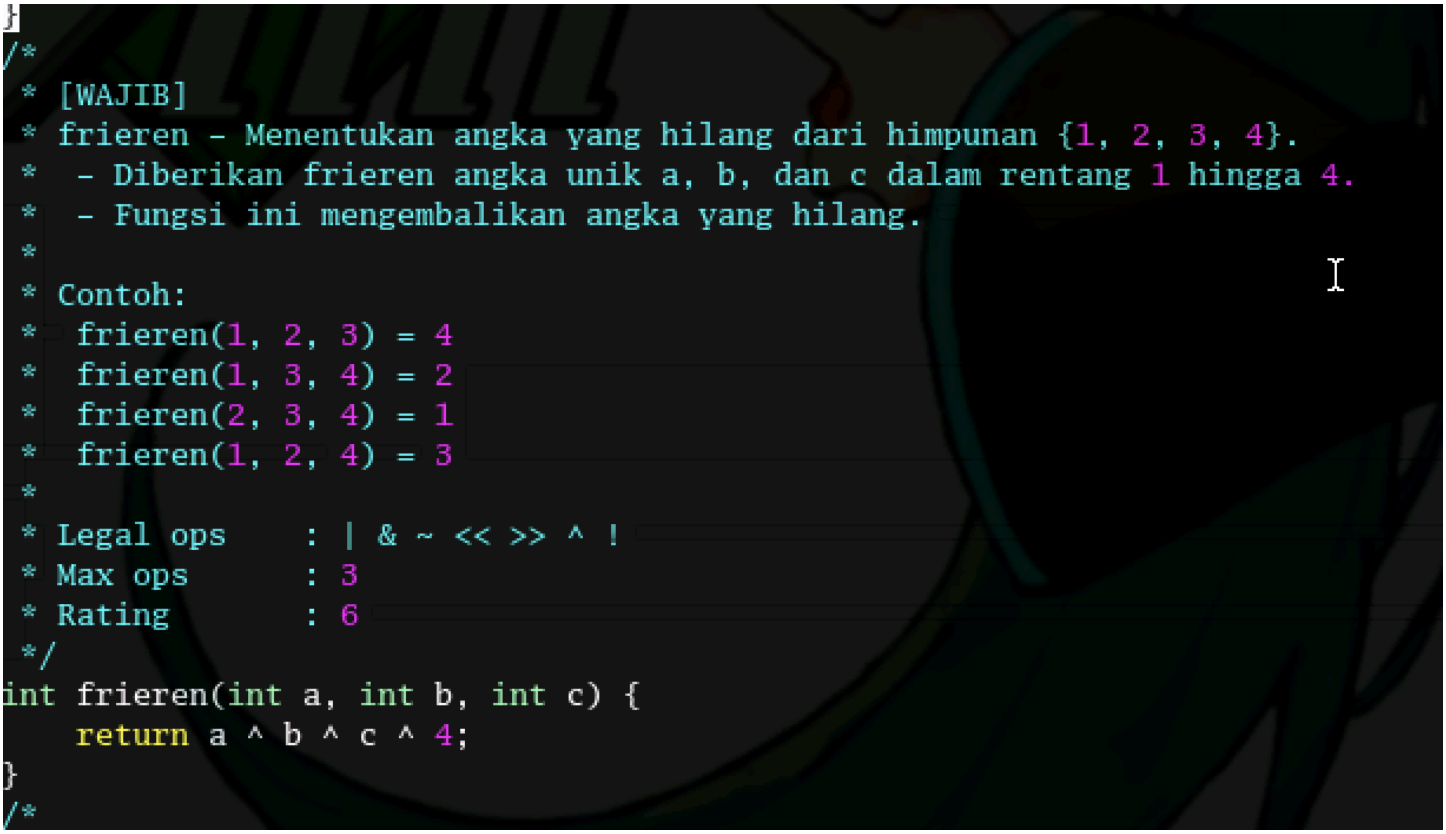
```
X & 15

X     : …XXXXXX
15    : …000111
————————————————
      : …000XXX
(x di antara [0, 1])
```

Karena yang diminta adalah 1 ketika dapat dibagi, maka kita harus negasi hasilnya, sekaligus merubah menjadi boolean. Dengan rumus ini,

```
!(x & 15)
```

Jika hasil operasi and terdapat sisa, kembalikkan 0.
Jika hasil operasi and tidak terdapat sisa, kembalikan 1.

3. Frieren



```
}
/*
 * [WAJIB]
 * frieren - Menentukan angka yang hilang dari himpunan {1, 2, 3, 4}.
 *  - Diberikan frieren angka unik a, b, dan c dalam rentang 1 hingga 4.
 *  - Fungsi ini mengembalikan angka yang hilang.
 *
 * Contoh:
 *  frieren(1, 2, 3) = 4
 *  frieren(1, 3, 4) = 2
 *  frieren(2, 3, 4) = 1
 *  frieren(1, 2, 4) = 3
 *
 * Legal ops    : | & ~ << >> ^ !
 * Max ops      : 3
 * Rating       : 6
 */
int frieren(int a, int b, int c) {
    return a ^ b ^ c ^ 4;
}
/*
```

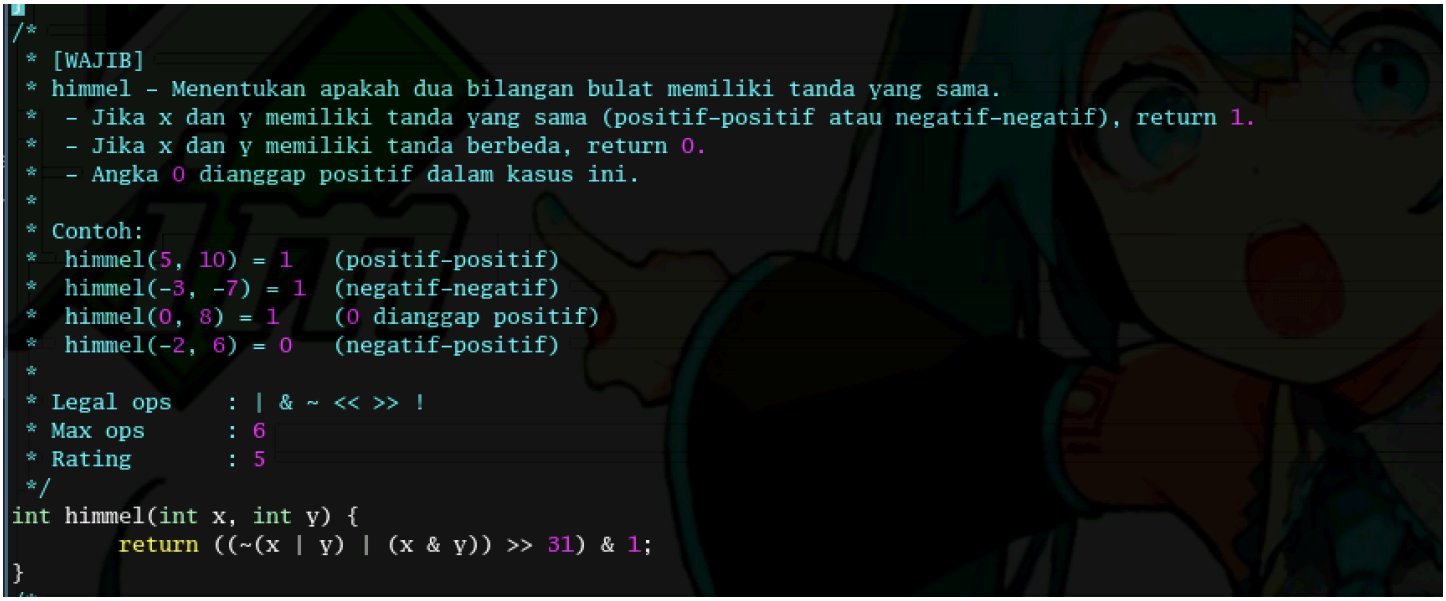
Di sini, kita diminta untuk mengembalikkan angka yang tidak ada dalam himpunan hanya dengan operasi bitwise dan memori konstan. Kita dapat menggunakan sifat spesial pada xor (^), yaitu

x ^ x = 0,

Atau, ketika angka yang sama dioperasikan xor, hasilnya pasti 0. Untuk kasus ini, dengan banyak angka, kita dapat membuat konstanta yang merupakan hasil operasi xor dari seluruh bilangan himpunan yang karena sifat tersebut, ketika dioperasikan xor oleh suatu elemen, bagian dari elemen tersebut akan hilang sehingga ketika semua a, b, c dioperasikan, yang tersisa pasti angka yang hilang.

```
a ^ b ^ c ^ 4
```

4. Himmel



```
/*
 * [WAJIB]
 * himmel - Menentukan apakah dua bilangan bulat memiliki tanda yang sama.
 *  - Jika x dan y memiliki tanda yang sama (positif-positif atau negatif-negatif), return 1.
 *  - Jika x dan y memiliki tanda berbeda, return 0.
 *  - Angka 0 dianggap positif dalam kasus ini.
 *
 * Contoh:
 *  himmel(5, 10) = 1   (positif-positif)
 *  himmel(-3, -7) = 1  (negatif-negatif)
 *  himmel(0, 8) = 1    (0 dianggap positif)
 *  himmel(-2, 6) = 0   (negatif-positif)
 *
 * Legal ops    : | & ~ << >> !
 * Max ops      : 6
 * Rating       : 5
 */
int himmel(int x, int y) {
        return ((~(x | y) | (x & y)) >> 31) & 1;
}
```

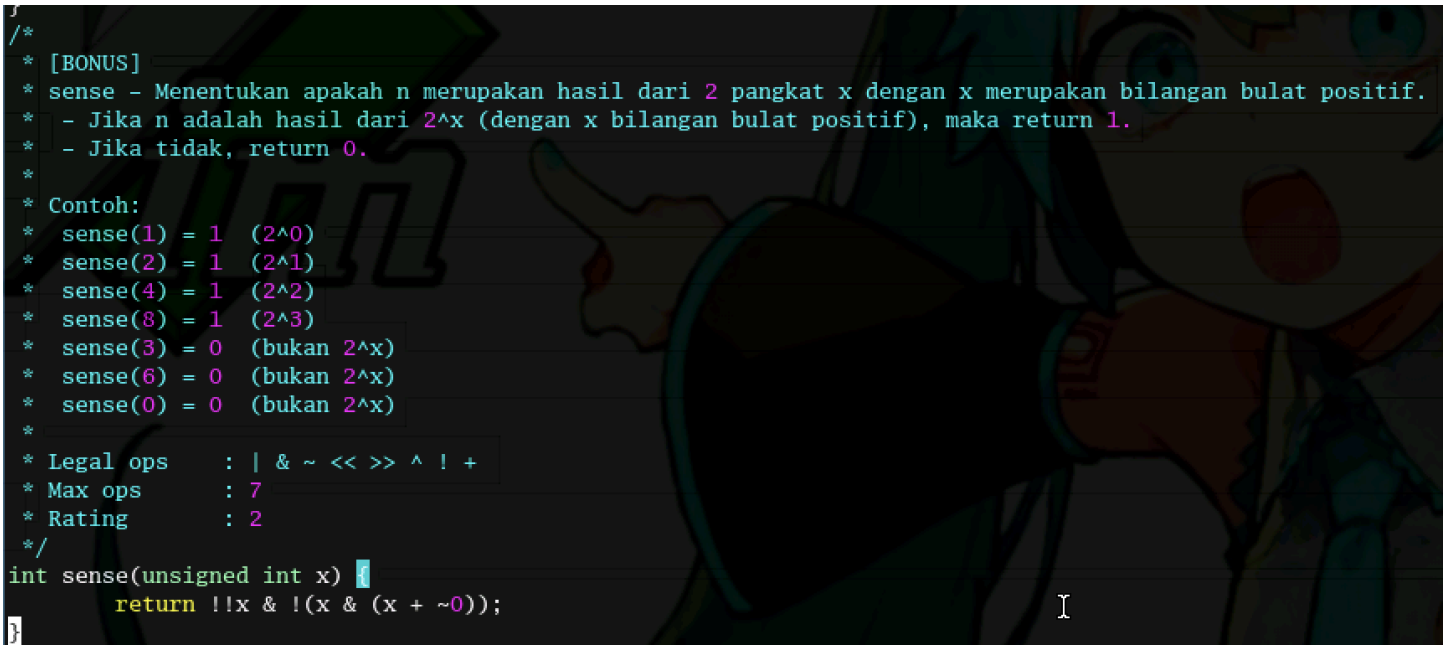
Kita diminta untuk menentukan kesamaan tanda (sign) pada dua bilangan, atau dengan kata lain biimplikasi dari tanda. Operasi biimplikasi sama saja dengan reverse xor, yaitu

```
~(x | y) | (x & y)
```

Setelah didapatkan hasil biimplikasi pada bit ke-31, kita perlu memindahkannya kepada bit ke-0 menggunakan right shift (>>) dan bitwise and (&) karena ketika bitnya 1 akan ke-extend menjadi “1111….”

```
((~(x | y) | (x & y)) >> 31) & 1
```

5. Sense [Bonus]



```
/*
 * [BONUS]
 * sense - Menentukan apakah n merupakan hasil dari 2 pangkat x dengan x merupakan bilangan bulat positif.
 *   - Jika n adalah hasil dari 2^x (dengan x bilangan bulat positif), maka return 1.
 *   - Jika tidak, return 0.
 *
 * Contoh:
 *   sense(1) = 1  (2^0)
 *   sense(2) = 1  (2^1)
 *   sense(4) = 1  (2^2)
 *   sense(8) = 1  (2^3)
 *   sense(3) = 0  (bukan 2^x)
 *   sense(6) = 0  (bukan 2^x)
 *   sense(0) = 0  (bukan 2^x)
 *
 * Legal ops    : | & ~ << >> ^ ! +
 * Max ops      : 7
 * Rating       : 2
 */
int sense(unsigned int x) {
        return !!x & !(x & (x + ~0));
}
```

Pada soal ini, diminta untuk menentukan apakah bilangan merupakan hasil dari 2^a, alias memiliki satu bit yang bernilai 1. Kita dapat menggunakan trik berikut yang dapat menghilangkan bit 1 paling kanan.

```
(x & (x - 1))
```

```
x     : ...XX1000
x - 1 : ...XX0111
----------------- &
      : ...XX0000
```

Operasi pengurangan akan mengubah seluruh 0 yang bersebelahan dari kanan ke kiri sehingga mencapai 1, yang ketika dioperasikan oleh bitwise and (&) akan, secara efektif, menghilangkan bit 1 paling kanan. Artinya, jika dia memiliki satu bit 1, hasilnya 0 karena satu-satunya bit hilang. Jika dia memiliki bit 1 lebih dari satu, hasilnya memiliki nilai. Akan tetapi, tanda '-' bukan operator legal. Maka, kita harus mengubahnya dengan

```
(x & (x + ~0))
```

karena bitwise not (~) dari 0 ekuivalen dengan -1. Karena yang diminta adalah boolean 1 jika dia merupakan hasil pangkat 2^a, maka diperlukan tanda negasi.

```
!(x & (x + ~0))
```

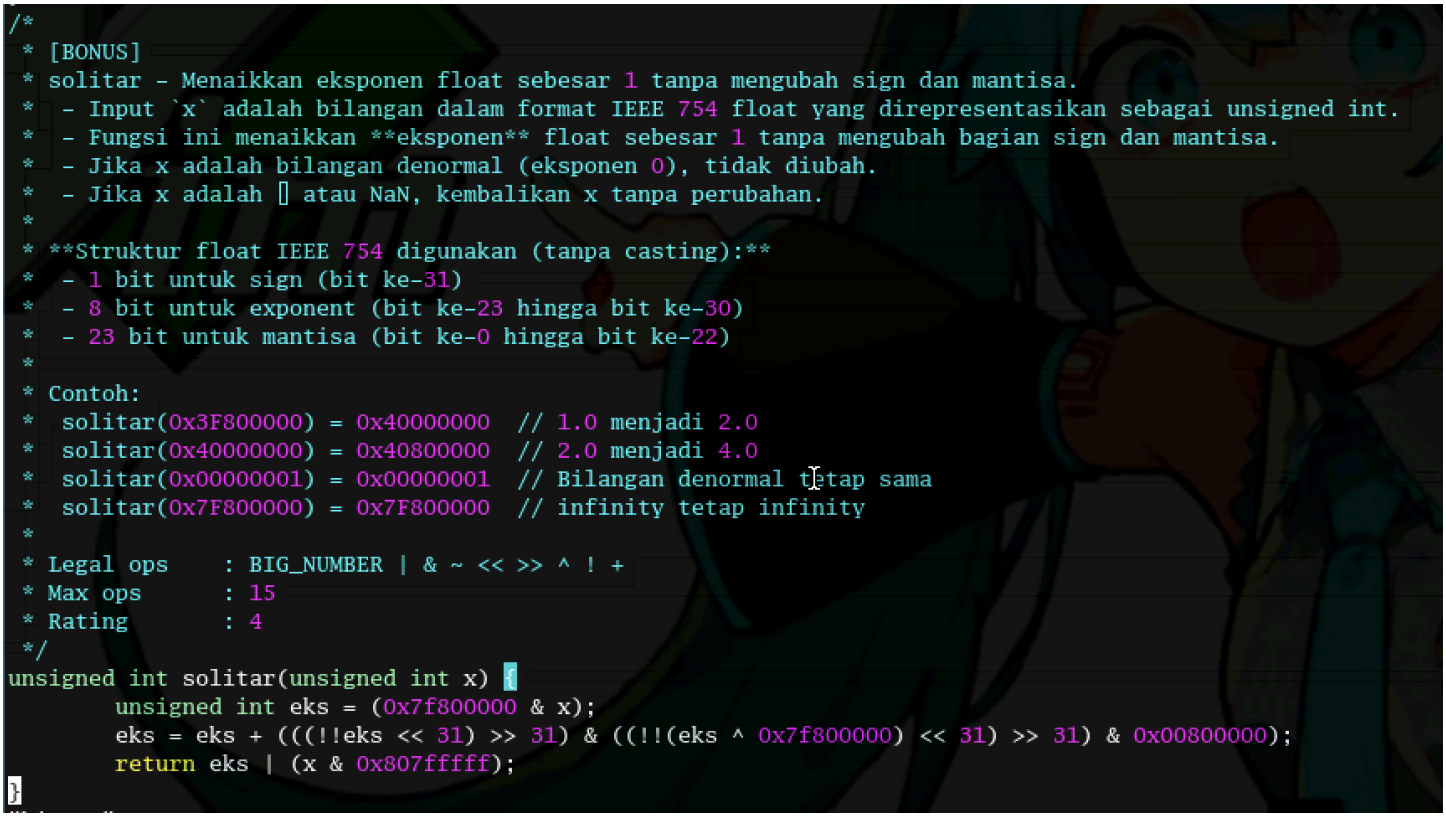
Selain itu kita harus menangani kasus ketika x adalah 0, hasil dari operasi tersebut akan bernilai 1, padahal bukan hasil pangkat 2^a. Maka, ditambahkan kondisi untuk mengecek apakah x memiliki nilai atau tidak dengan

```
!!x
```

dimana negasi dua kali berlaku sebagai casting ke boolean. Didapatkan,

```
!!x & (x & (x + ~0))
```

6. Solitar [Bonus]



```
/*
 * [BONUS]
 * solitar - Menaikkan eksponen float sebesar 1 tanpa mengubah sign dan mantisa.
 *   - Input `x` adalah bilangan dalam format IEEE 754 float yang direpresentasikan sebagai unsigned int.
 *   - Fungsi ini menaikkan **eksponen** float sebesar 1 tanpa mengubah bagian sign dan mantisa.
 *   - Jika x adalah bilangan denormal (eksponen 0), tidak diubah.
 *   - Jika x adalah [] atau NaN, kembalikan x tanpa perubahan.
 *
 * **Struktur float IEEE 754 digunakan (tanpa casting):**
 *   - 1 bit untuk sign (bit ke-31)
 *   - 8 bit untuk exponent (bit ke-23 hingga bit ke-30)
 *   - 23 bit untuk mantisa (bit ke-0 hingga bit ke-22)
 *
 * Contoh:
 *   solitar(0x3F800000) = 0x40000000  // 1.0 menjadi 2.0
 *   solitar(0x40000000) = 0x40800000  // 2.0 menjadi 4.0
 *   solitar(0x00000001) = 0x00000001  // Bilangan denormal tetap sama
 *   solitar(0x7F800000) = 0x7F800000  // infinity tetap infinity
 *
 * Legal ops    : BIG_NUMBER | & ~ << >> ^ ! +
 * Max ops      : 15
 * Rating       : 4
 */
unsigned int solitar(unsigned int x) {
        unsigned int eks = (0x7f800000 & x);
        eks = eks + ((((!!eks << 31) >> 31) & ((!!(eks ^ 0x7f800000) << 31) >> 31) & 0x00800000);
        return eks | (x & 0x807fffff);
}
```

Kita diminta untuk menambahkan eksponen sebanyak 1, kecuali jika x adalah bilangan denormal (eks = 0) atau infinity (eks = 255, angka dapat ditebak dari contoh). Selain itu, tidak ada logika tambahan karena ketika eksponen maksimal, dia tidak berubah.

Pertama, kita perlu mengambil bagian eksponennya saja, ini dapat dilakukan dengan mengoperasikan x dengan bit flag konstan yang menduduki bit ke-23 hingga bit ke-30,

```
0x7f800000:     01111111100000000000000000000000
```

Lalu, tambahkan eks dengan 1 jika memenuhi kondisi. Sifat kondisional ini dapat dicapai dengan membuat bit flag penuh dari boolean dan mengoperasikannya dengan bilangan yang bergantung pada kondisional tersebut. Boolean dapat diubah menjadi bit flag penuh menggunakan left shift (<<) lalu right shift (>>). Jika b == 0, bit flag tetap kosong. Jika b == 1, bit flag menjadi penuh (1111...).

```
((boolean << 31) >> 31)
```

Dengan mengubah kedua kondisi tadi menjadi boolean akan mendapatkan,

```
((!eks << 31) >> 31), eks != 0
```

```
((!!(eks ^ 0x7f800000) << 31) >> 31), eks != 255 (pada bit ke-23)
```

Gabungkan seluruh kondisional itu dengan penambahan didapatkan,

```
Eks = eks + (((!eks << 31) >> 31) & ((!!(eks ^ 0x7f800000) << 31) >> 31) & 0x00800000)
```

dimana 0x00800000 adalah 1 pada bit ke-23.

Terakhir gabungkan hasil penjumlahan tersebut dengan signed dan mantissa x sehingga

```
eks | (x & 0x807fffff)
```

*Ngl, saat menulis ini, aku masih gatau mantissa itu apa dan gimana cara ngitung float tapi dari soalnya bisa ditebak, so yeah.

Thank you for reading!!!