

1. როგორც ვიცით XOR ოპერატორი აბრუნებს true-ს მხოლოდ მაშინ, როცა მნიშვნელობები განსხვავდება.

```
true ^ false = true
false ^ true = true
true ^ true = false
false ^ false = false
```

Negative XOR - ის შემთხვევაში ეს პირობები პირიქითაა, ანუ true ბრუნდება მხოლოდ მაშინ, როცა მნიშვნელობები ერთი და იგივეა.

```
true ^ false = false
false ^ true = false
true ^ true = true
false ^ false = true
```

აქედან გამომდინარე თუ ჩვენი მიზანია გადავიყვანოთ მოცემული მაგალითი ( **$h \geq 180 \wedge hc == 'b'$** ) Negative XOR-ში, შეგვიძლია გამოვიყენოთ ! ოპერატორი:

**$!(h \geq 180 \wedge hc == 'b')$**  - ამგვარად XOR ოპერატორი დააბრუნებს საპირისპირო მნიშვნელობებს.

2. პირველ რიგში გავამარტივოთ მოცემული ცვლადი **`bool isOk = h >= 180 & hc == 'b';`**

შეგვიძლია მაგალითისთვის შემოვიტანოთ:

```
bool a = (h >= 180);
bool b = (hc == 'b');
```

გამარტივებული სახით:

```
isOk = (a & b);
```

მოცემული გამარტივებული ცვლადი შეგვიძლია განვიხილოთ დე მორგანის კანონის ფარგლებში, რადგან ჩვენი მიზანია ჩავანაცვლოთ & ოპერატორი | და ! ოპერატორებით.

დე მორგანის წესის მიხედვით:

```
(a & b) = !(a | b);
```

აქედან გამომდინარე შეგვიძლია isOk ჩავწეროთ შემდეგი სახით:

```
isOk = !(h >= 180 | hc == 'b');
```

მოცემულ შემთხვევაში თუ პირველი (a) პირობა შესრულდება და დაბრუნდება **true**

**!** ოპერატორი შეაბრუნებს და დაგვრჩება **false**. იგივე მოხდება მეორე (b) პირობის შესრულებისას და გვექნება შემდეგი მოცემულობა:

```
isOk = !(false | false);
```

**(false | false)** დააბრუნებს false-ს, **!** ოპერატორი შეაბრუნებს და გამოვა, რომ **isOk = true**;

იგივე პასუხი გვექნებოდა პირველ შემთხვევაშიც, რადგან ორივე პირობა შესრულდებოდა და **true & true** დააბრუნებდა true-ს.

3. ამ პირობის ჩასაწერად შეგვიძლია ავიღოთ რამდენიმე ცვლადი:

```
bool tall = true;  
bool brunette = true;
```

თუ გვინდა, რომ ამოვარჩიოთ ადამიანები, რომლებიც არ არიან მალელები და შავგრემნები ერთდროულად უნდა შესრულდეს შემდეგი პირობა:

```
!(tall & brunette)
```

დე მორგანის კანონის მიხედვით **!(a & b) = (!a | !b)**. აქედან გამომდინარე:

```
!(tall & brunette) = (!tall | !brunette);
```

4. რადგან **|** არის **bitwise** ოპერატორი შეგვიძლია **h** განვიხილოთ კონკრეტულ მაგალითზე:

```
h = 0b_0000_0101;
```

**|** ოპერატორის შემთხვევაში operand-ების 0-ები ნაცვლდება იგივე ადგილას მყოფი 1-ებით, ანუ **h | h** შემთხვევა კონკრეტულ მაგალითზე იგივეა რაც:

**0000\_0101 | 0000\_0101** რაც მოგვცემს **0000\_0101**-ს და **h**-ის ნებისმიერი მნიშვნელობისთვის გამოვა, რომ **h | h = h**.

-----

**h | 0** შემთხვევაში ბიტების იგივე მაგალითზე გამოვა შემდეგი ტოლობა:

**0000\_0101 | 0000\_0000** რაც მოგვცემს **h**-ის მნიშვნელობას, **0000\_0101**-ს, ანუ **h | 0 = h**.

-----

**h | 1** შემთხვევაში შედეგი დამოკიდებულია **h**-ის მნიშვნელობაზე. მაგალითად თუ ავიღებთ კენტ რიცხვს:

**0000\_0101 | 0000\_0001** - ამ შემთხვევაში დაგვიბრუნდება **h**-ის მნიშვნელობა **0000\_0101**

თუ ავიღებთ ლუწ რიცხვს:

**0000\_1010 | 0000\_0001** - ამ შემთხვევაში დაგვიბრუნდება  **$h + 1$**  მნიშვნელობა, ანუ **0000\_1011**

თუ  **$h$**  არის ლუწი  **$h | 1 = h + 1$**

თუ  **$h$**  არის კენტი  **$h | 1 = h$**

-----

როგორც ვიცით  $\sim$  ოპერატორი 0 და 1-ს ადგილებს უცვლის, მაგალითად თუ მოცემული გვაქვს:

**$h = 0000\_0101$ ;**

**$\sim h = 1111\_1010$ ;** რაც ავტომატურად გულისხმობს, რომ  **$\sim h = -h - 1$ ;**

**1111\_1111**

| ოპერატორის შემთხვევაში  **$h$**  და  **$\sim h$**  მნიშვნელობები „შეიკრიბება“ და გამოვა:

**$h - h - 1$**

ანუ  **$h$** -ის ნებისმიერი მნიშვნელობისთვის  **$h | \sim h = -1$ .**

5. ზემოთ განხილული მაგალითების მიხედვით შეგვიძლია განვიხილოთ & ოპერატორიც:

**$h = 0000\_0101$**

**0000\_0101**

**0000\_0101**

& ოპერატორის შემთხვევაში გადმოდის 1-ები, რომლებიც განლაგებულია ორივე ცვლადში, ანუ ამ შემთხვევაში გამოვა **0000\_0101** ანუ  **$h$** . აქედან გამომდინარე  **$h$** -ის ნებისმიერი მნიშვნელობისთვის  **$h \& h = h$ ;**

-----

**$h \& 0$**  -ის შემთხვევაში ყველა ბიტის ადგილს დაიკავებს 0-ის ბიტები (**0000\_0000**) და  **$h$** -ის ნებისმიერი მნიშვნელობისთვის გამოვა, რომ  **$h \& 0 = 0$ ;**

-----

**$h \& 1$**  -ის შემთხვევაში შედეგი დამოკიდებულია  **$h$**  კენტია თუ ლუწი. თუ ავიღებთ კენტ რიცხვს:

**0000\_0101 | 0000\_0001** - ამ შემთხვევაში დაგვიბრუნდება **0000\_0001** ანუ  **$h \& 1 = 1$** , თუმცა ლუწის შემთხვევაში:

**0000\_1010 | 0000\_0001** - დაგვიბრუნდება **0000\_0000** ანუ  **$h \& 1 = 0$**

თუ  **$h$**  კენტია  **$h \& 1 = 1$**

თუ  **$h$**  ლუწია  **$h \& 1 = 0$**

-----

**$h \& \sim h$**  -ის შემთხვევაში გამოვა, რომ:

**$h = 0000_0101$**

**$\sim h = 1111_1010$**

და ყველა ბიტი 0-ზე დაჯდება, აქედან გამომდინარე  **$h$** -ის ნებისმიერი მნიშვნელობისთვის  **$h \& \sim h = 0$** ;