STMC HKOI Training

Mathematical Foundations

Chan Yan Mong

October 10, 2021

Exponent Notation

- Many times in mathematics we encounter expressions that involve repeated multiplications of same number
- E.g. $2 \times 2 \times 2 \times 2$; $\overbrace{4 \times 4 \cdots \times 4}^{51 \text{ times}}$
- To simply our notation, we will introduce the **exponent notation**

Exponent Notation

Definition (Positive Exponent)

Let $N \in \mathbb{Z}^+$ and $m \in \mathbb{R}$. The N th power of m, denoted m^N, is defined as:

$$m^N = \overbrace{m \times m \times \cdots \times m \times m}^{N \text{ times}} \tag{1}$$

For example, in previous examples:

$2 \times 2 \times 2 \times 2 = 2^4$ and $\overbrace{4 \times 4 \cdots \times 4}^{51 \text{ times}} = 4^{51}$

Properties of exponent

It’s easy to proof that the exponent satisfy the following properites:

Theorem (Properties of exponent)

1. $a^m a^n = a^{m+n} \quad m, n \in \mathbb{Z}^+$
2. $a^m / a^n = a^{m-n} \quad m, n \in \mathbb{Z}^+, m > n$
3. $(a^m)^n = a^{mn} \quad m, n \in \mathbb{Z}^+$

Proof.

Omitted, explain in class $\square$

Extending exponent

- To make the theorems above more useful, we define extend our definition of exponent in the following way:

Definition (Negative and Zero exponent)

Let $N \in \mathbb{Z}$ and $m \in \mathbb{R}$. The N th power of m, denoted m^N, is defined as:

$$m^N = \begin{cases} \overbrace{m \times m \times \cdots \times m \times m}^{N\text{ times}}, & N > 0 \\ 1, & N = 0 \\ 1/m^{|N|}, & N < 0 \end{cases} \tag{2}$$

Extending exponent

Examples:

- $2^0 = 1$
- $6^{-3} = 1/6^3$

Such notation is very useful. Consider the following expression:

$$\begin{aligned} S &= \frac{1}{2} + \frac{1}{6} + \frac{1}{16} \\ &= 2^{-1} + \left(2^{-1}\right)\left(3^{-1}\right) + 2^{-4} \\ &= \left(2^{-4}\right)\left(3^{-1}\right)\left[\left(2^{3}\right)\left(3^{1}\right) + 2^{3} + 3^{1}\right] \\ &= \frac{35}{48} \end{aligned}$$

Fractional exponent

- In a similar vein, we can define fractional exponent using the multiplication rule:

$$\left(m^{1/N}\right)^N = m^{N/N} = m$$

Hence:

$$m^{1/N} = \sqrt[N]{m}$$

Fractional exponent

Definition (Fractional exponent)

Let $m \in \mathbb{R}$ and $m > 0$ and $N \in \mathbb{Z}$, then we define the fractional exponent of m as:

$$m^{1/N} = \sqrt[N]{m} \tag{3}$$

where $\sqrt[N]{\cdots}$ denotes the N-th root of the number

Loop: Repeat and repeat ….

- From the examples above, we see the a looping structure always consist of two parts:
  1. The code inside the code that is looped over
  2. A condition that is checked everytime the loop ran to decide whether the loop should continue
- Example:
  - Recieving user input (code inside loop); Is the answer right (terminate condition)
  - Reading files (code inside loop); Is the end of file reached (terminate condition)
  - Main game code (code inside the loop); Is the game over (terminate condition)
  - Searching for answers (code inside the loop); Is the solution found (terminate condition)

Types of loop

- Loops can also be classified into two categories:
  1. **Pre-check loops** and
  2. **Post-check loops**
- In this lesson we will focus on post-check loops and deal with pre-check loops next lesson

`while` loop

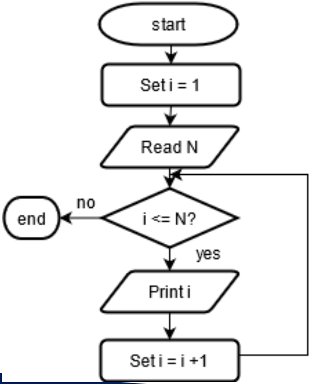
Example: Print first N positive integers

- Suppose you want to write a program that takes in an integer N and print out all positive integers $\leq N$. We can implement that using while loop.
- For example:

```
$./main          $./main          $./main
5                4                100
1                1                1
2                2                2
3                3                .... /* too long won't list here*/
4                4                99
5                                 100
```

`while` loop

- Let's look at the flowchart of the program
- Here the looping condition is $i \leq N$ and the main code that is looped is "Print i" and "Set i = i + 1"
- Notice "Set i = i + 1" is crucial otherwise i will always be smaller than N. This will cause an **infinite loop**



while loop

- Let's implement the code in C/C++
- Notice when the condition is true, it will loop

```
int i = 1, N;
scanf("%d",&N);

/* The while loop */
while(i <= N) {      // Is i <= N? If true then loop
  printf("%d\n",i); // Print i
  i = i + 1;        // Set i = i + 1
}
```

`while` loop

Example: Sum of first n cubes

Write a program that takes n as an input and compute the sum of first n cubes S_n:

$$S_1 = 1^3$$

$$S_2 = 1 + 2^3$$

$$\cdots$$

$$S_n = 1^3 + 2^3 + \cdots + (n-1)^3 + n^3$$

while loop

- This is similar to our previous example:

```c
int i = 1, sum = 0, N;
scanf("%d",&N);

/* The while loop */
while(i <= N) {      // Is i <= N? If true then loop
  sum = sum + i*i*i;
  i = i + 1;
}

printf("Required sum: %d\n",sum);
```

`while` loop

Example: Input validation

- Suppose we are writing a program that computes the BMI of a student given weight and height as input using the formula BMI $= \text{weight}/\text{height}^2$
- Naturally, we will do something similar to this to get the input:

```c
float height, weight;
scanf("%f %f",&height,&weight);
```

`while` loop

- Although we know that weight and height must be larger than zero and they both have an upper bound. There are nothing stopping the user from entering the following in the program:

```
./main
-100 1000000 /* People with -100m and 1000000kg ???? */
```

- Therefore, we need a way to check and prevent these things from happening
- Write a program that check if $0 \leq$ weight ≤ 600kg and $0 \leq$ height ≤ 3m and keep asking the user to reenter weight and height until the correct value is recieved.

`while` loop

- Here is the code:

```c
float height, weight;
printf("Please enter height and weight: ");
scanf("%f %f",&height, &weight);

while(!(0.0 <= weight && weight <= 600.0 && 0.0 <= height && height <= 3.0)) {
  printf("Height must be between 0 to 3.0 and weight must be between 0 to 600.0!\n");
  printf("Re-enter height and weight: ");
  scanf("%f %f",&height,&weight);
}

printf("Height, weight and BMI = %f, %f, %f\n",height,weight,height/(weight*weight));
```

`while` loop

Examples: Fixed point iteration

- Let x, a be two non negative real numbers. If $x^2 = a$, then x is called the square root of a and we denote $x = \sqrt{a}$
- For example, $3^2 = 9$ so 3 is the square root of 9. We also denote $3 = \sqrt{9}$
- Now, how can we find interesting

`if-else` statement

- This can be done by the `if-else` statement in C/C++
- We ignore the `main()` part and focus on the if statement itself

```
    /* This is inside main(); I am just lazy */

    if(/* Is it raning? */){
      /* Stay at home */
    } else {
      /* Go out */
    }

```

Filling the condition

- But how do we fill in the `/* Is it raining ? */` part?
- In general, we fill that part with an **boolean expression** that return *true* or *false* upon evaluation
- For example of boolean expressions are:
  - Is x > y ?
  - Is x equal to y?
  - etc.
- Let's look at some examples to see how exactly can we do that in C/C++

Equal to ==

- The operator == is the operator for **equal to**
- **Do not confuse it with a single =**
- `x==y` checks if x is equal to y
- An example of the so called *comparsion operators*
- Examples:
  - `1 == 0` $\rightarrow$ `false`
  - `3 == 3` $\rightarrow$ `true`
  - `'a' == 'A'` $\rightarrow$ `false`
  - `'b' == 'b'` $\rightarrow$ `true`

Equal to ==

Example:

John is a middle schooler. Everyday he can either be happy or unhappy. If he is happy, he will study; If he is not, he will play computer games. Let `happiness` be John's happiness. If `happiness` is 1, he is happy; otherwise, he is not. Write a program that predicts what John will do given his happiness.

Input: An integer `happiness` which is either 0 or 1
Output: Print ”He will study” if he is happy and ”He will play computer games” if not.

Equal to ==

```c
#include <stdio.h>

int main() {
  int happiness;
  scanf("%d",&happiness); // Read happiness

  if(happiness == 1) {    // If happiness equal to 1
    printf("He will study\n");
  } else {
    printf("He will play computer games\n");
  }

  return 0;
}
```

Equal to ==

Example: Odd or even

Write a program that tells you whether an integer is odd or even. Your program should take in an integer `num` and return `ODD` if it's odd and `EVEN` otherwise. You are given that `num` > 0.

Example Input/output

```
Example 1:     Example 2:     Example 3:     Example 4:
$./main        $./main        $./main        $./main
1              5              6              10
ODD            ODD            EVEN           EVEN
```

Larger than > or smaller than <

- Similarly, we have `x > y` and `x < y`
- Checks if x is strictly larger than y or strictly smaller than y
- Example:
  - `1.2 < 3.7` → `true`
  - `0 < 1` → `false`
  - `3 > 1` → `true`
  - `9.9 > 3.7` → `true`
  - `3 < 3` → `false`

Larger than > or smaller than <

Example: Amber rainstorm signal

According to HKO, the Black rainstorm signal is issued if the hourly rainfall exceeds 70mm. Write a program that takes in the hourly rainfall `rainfall` in mm and determine whether the black rainstorm signal is issued. Return `BLACK` if so and `OTHERS` if otherwise.

Source: HKO

Example Input/output

```
Example 1:     Example 2:     Example 3:     Example 4:
$./main        $./main        $./main        $./main
0              70.0           72.4           23.1
OTHERS         OTHERS         BLACK          OTHERS
```

Larger than > or smaller than <

Exampe: Overbudget

Merry has $300 dollar in her pocket. Since Christmas is approaching, she decided to buy some gifts for her friends. The types of gifts she wanted to buy are: Pencil ($3.0 each), Cake ($11.0 each) and Book ($ 80.0 each). Suppose she bought n_p pencil, n_c cake and n_b books. Write a program to determine whether she exceeded her budget. If no, print NO OVERBUDGET; otherwise, print EXCEEDED <amount>.

```
Example 1:            Example 2:         Example 3:
$ ./main              ./main             ./main
101 0 0               1 27 0             5 4 3
EXCEEDED 3.000000     NO OVERBUDGET      NO OVERBUDGET
```

More operators: >=, <=, !=

- Similarly, we also have **smaller than or equal to** <= and **larger than or equal to** >=
- Examples
  - `1 >= 1` → `true`
  - `3 >= 1` → `true`
  - `4 <= 1` → `false`
- We also have **not equal to** !=
- Examples:
  - `1 != 1` → `false`
  - `3 != 2` → `true`
  - `'a' != 'a'` → `false`
  - `'a' != 'A'` → `true`

More decisions `if`, `else if`, `else`

- We can make more decisions by `else if` statement

```
if(/* condition 1*/) {

  // Run if condition 1 is true

} else if (/* condition 2 */) { // Check cond 2 if cond 1 is false

  // Run if conditional 2 is true

} else if (/* condition 3 */) { // Check cond 3 if cond 1,2 are
  false

  // Run if conditional 3 is true

} else {

  // Run if cond 1,2,3 all false

}
```

More decisions

Example: Rainstorm signal+

According to HKO, the amber, red and black rainstorm signal is issued if the hourly rainfall exceed 30mm, 50mm and 70mm respectively (inclusive). Write a program that takes in a float `rainfall` and return `AMBER`, `RED` or `BLACK` accordingly if there's a signal, and `NO SIGNAL` if there is no signal. Furthermore, return `ERROR` if `rainfall` < 0.

```
Example 1:        Example 2:        Example 3:        Example 4:
$./main           $./main           $./main           $./main
0                 70.0              53.4              30.2
NO SIGNAL         BLACK             RED               AMBER

Example 5:
$./main
-3
ERROR
```

Boolean expression

- Recall **boolean expression** is an expression that **either returns true or false**
- Examples:
  - ”John has beard”
  - ”Spiders more than 2 legs”
  - ”x is equal to y”
  - ”There is more sand on the Earth than stars on the universe”
- Now we want to *combine* or *modify* these expressions

Combining expressions

- Let's consider how boolean expressions can be combined
- Consider the statements:
  1. Today is raining
  2. Eva has an umbrella
- One way to combine them is to use the connective ”and”
- So we have ”Today is raining and Eva has an umbrella”
- Now, when is the new statement true?

Combining expression

We can investigate the problem by using a **truth table**

Today is raining	Eva has an umbrella	Today is raining and Eva has an umbrella
T	T	T
T	F	F
F	T	F
F	F	F

We can see that the final statement ”Today is raining and Eva has an umbrella” is true only if *both* ”Today is raining” and ”Eva has an umbrella” are individually true

Combining expression

- In fact, we can see the above table is not limited to ”Today is raining” and ”Eva has an umbrella”
- For any boolean expression *a*, *b*, we can always combine *a*,*b* by asking if *a* and *b* is true
- Hence, the truth table above define the operation ”and”
- Let’s define different operations together

Logical AND $\wedge$

- The **logical AND** $\wedge$ is a binary operation
- It combines two statements a, b and return true only if *both* statements are true
- In C/C++ this is done using the && operator
- Example:
  - `1 <= var && var < 3`
  - `(number % 3 == 0) && (number % 2 != 0)`
  - `(chr != 'A') && (chr != 'B')`

a	b	$a \wedge b$
T	T	T
F	F	F
F	T	F
T	F	F

Table 1: Truth table of $\wedge$

Logical AND $\wedge$

Exercises

1. Write a program that takes in an integer `num` and print `"DIV BY 6"` if it's divisble by 6 and `"NOT DIV BY 6"` otherwise
2. Write a program that takes in a float `temp` and check if $0 \leq \text{temp} < 100.0$. Print `"YES"` if it's within the range and `"NO"` otherwise

Logical OR $\vee$

- The **logical OR** $\vee$ is a binary operation
- It combines two statements a, b and return true only if *either* of the statements are true
- In C/C++, this is done using `||` operator
- Example:
  - `(var == 1) || (var == 2)`
  - `!(count > 1 || count < -1)`
  - `(var == 1) || (var != 3)`

a	b	$a \vee b$
T	T	T
F	F	F
F	T	T
T	F	T

Table 2: Truth table of $\vee$

Logical OR $\vee$

Exercises

1. Write a program that takes in a number `num` and print `YES` if it is divisible by 2 or 3 and `NO` if otherwise
2. Body temperature T is considered `NORMAL` if $36 \leq T \leq 38$ and `ABNORMAL` otherwise. Without using `&&`, write a program that takes in a float `temp` and checks whether the body temperature is `NORMAL` or `ABNORMAL`

Logical OR $\vee$

Exercise

3. a, b are numbers that can only be 0 or 1. The operation $a \oplus b$ is defined using the table below. Write a program that takes in two integer `a`, `b` as input and compute $a \oplus b$

a	b	$a \oplus b$
0	1	1
1	0	1
0	0	0
1	1	0

4.

Negation operation $\neg$

- The **negation operation** $\neg$ is a unitary operation
- It is equivalent to adding ”not” to the statement
- In C/C++, this is done by adding `!` in front of conditionals
- Example:
  - `!(var > 3)` $\leftrightarrow$ `(var <= 3)`
  - `!(var == 3)` $\leftrightarrow$ `(var != 3)`

a	$\neg a$
T	F
F	T

Table 3: Truth table of $\neg$

Summary Exercises

1. Write a program that reads in 3 numbers and ouput the largest number. You are guaranteed that no two numbers are equal. Expected ouput

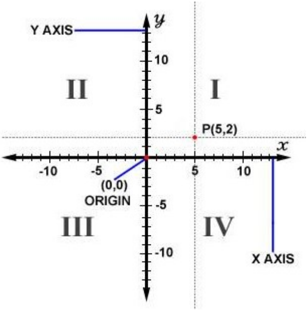
```
$./main
13 83 19
2nd number is largest. The value is 83

$./main
77 21 3
1st number is largest. The value is 77

$./main
0 21 88
3r number is largest. The value is 88
```

Summary Exercises

2. Write a program that takes a coordinate point in a XY coordinate system and determine in which quadrant the coordinate point lies. If either *x* or *y* is zero, output `UNDEFINED`. Refer to the figure if you don't know what is a quadrant.



Source: CSGNetwork

Summary Exercise

Expected output:

```
./main          ./main          ./main          ./main
1.3 4.5         -1.4 2.0        -3.2 -4.4       3.3 -3.1
Quadrant 1      Quadrant 2      Quadrant 3      Quadrant 4

./main          ./main          ./main
0 4.5           -3.3 0          0 0
UNDEFINED        UNDEFINED      UNDEFINED
```

Summary Exercise

3. Write a program to find the eligibility of admission for a professional course based on the following criteria:

   Eligibility Criteria : Marks in Maths >=65 and Marks in Phy >=55 and Marks in Chem>=50 and Total in all three subject >=190 or Total in Maths and Physics >=140

Summary Exercise

Expected output:

- Math = 17, Phy = 32, Chem = 1

```
./main
17 32 1
Not Eligible
```

- Math = 70, Phy = 65, Chem = 55

```
./main
70 65 55
Eligible
```

- Math = 70, Phy = 69, Chem = 99

```
./main
70 69 99
Eligible
```

Summary Exercise

4. Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are. Write a program that determines whether a year is a leap year.

```
$./main              ./main              ./main
1700                 1600                2012
Not leap year        Is leap year        Is leap year
```

Boolean arithmetic

- We have just looked at conditional statements that are connected using not, or, and operators
- In fact there are algebraic structure associated with these operators called **boolean arithmetic**
- Understanding these arithmetic rules can help us simply and rewrite our conditional statements

Properties of boolean operators

- Associativity
  1. $x \vee (y \vee z) = (x \vee y) \vee z$
  2. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
- Commutativity
  1. $x \vee y = y \vee x$
  2. $x \wedge y = y \wedge x$
- Distributive of $\wedge$ over $\vee$
  1. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
  2. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Properties of boolean operations

- Identities
  1. $x \vee F = x$
  2. $x \wedge T = x$
- Annihilators
  1. $x \vee T = T$
  2. $x \wedge F = F$
- Double negation
  1. $\neg(\neg x) = x$
- De Morgan's law
  1. $\neg(x \vee y) = \neg x \wedge \neg y$
  2. $\neg(x \wedge y) = \neg x \vee \neg y$

Exercise

Prove the following Identities:

1. $x \wedge (x \vee y) = x$
2. $\neg x \wedge y = \neg(\neg y) \wedge \neg x$
3. $a = b \wedge a$ if and only if $b = a \vee b$
4. Prove by the De Morgan's law by truth table
5. Prove by the distributive laws by truth table
6. Define $x \rightarrow y = \neg x \wedge y$. Show that $(x \wedge y) \rightarrow y = T$
7. Define $x \leftrightarrow y = (x \rightarrow y) \wedge (y \rightarrow x)$. Show that $x \leftrightarrow y = (x \wedge y) \vee (\neg x \wedge \neg y)$
8. Simplify$(x \vee y) \wedge \neg(\neg x \wedge y)$

Mathematical proving

- A **mathematical proof** is a sequence of logical statements, one implying another, which gives an explanation of why a given statement is true.
- Mathematical proof is *absolute*, which means that once a theorem is proved, it is proved for ever.

Mathematical proving

Example: Geometric sequence

One day, Judy is bored. So she tries to play around with sum of power of two. She tabulated her some an found out something interesting:

1	$2^1 = 2$	$1 + 2 = 3$
2	$2^2 = 4$	$1 + 2 + 2^2 = 7$
3	$2^3 = 8$	$1 + 2 + 2^2 + 2^3 = 15$
4	$2^4 = 16$	$1 + 2 + 2^2 + 2^3 + 2^4 = 31$
5	$2^5 = 32$	$1 + 2 + 2^2 + 2^3 + 2^4 + 2^5 = 63$

Mathematical proving

So she made the following hypothesis:

Hypothesis

Let n be an integer and $n \geq 1$. Then $1 + 2 + \cdots + 2^{n-1} = 2^n - 1$

Question: Is she correct?

Mathematical proving

In fact she IS correct. To see that, let S be the required sum:

$$S = 1 + 2 + 2^2 + \cdots + 2^{n-2} + 2^{n-1}$$

Then $2S$ is:

$$2S = 2 + 2^2 + 2^3 + \cdots + 2^{n-1} + 2^n$$

Mathematical proving

So:

$$\begin{aligned} S &= 2S - S \\ &= (2 + 2^2 + 2^3 + \cdots + 2^{n-1} + 2^n) - (1 + 2 + 2^2 + \cdots + 2^{n-2} + 2^{n-1}) \\ &= 2^n - 1 \end{aligned}$$

Which proves our hypothesis. Now, even if we don't list out every single number n, we will know the claim is true.

Mathematical proving

- This is an example of a proof
- In general, we need a proof in order to be certain something is true, because apparent patterns can and had failed in the past
- Example:
- **Claim:** $n^2 + n + 41$ is a prime number (which is false)

n	1	2	3	4	$\cdots$	38	39	40
$n^2 + n + 41$	43	47	53	61	$\cdots$	1523	1601	1681
Is prime?	Yes	Yes	Yes	Yes	$\cdots$	Yes	Yes	No

Mathematical proving

Exercise

- Prove that $n(n+1)$ is always divisible by 2
- In a group of 366 people, there must be at least two people with the same birthday
- Prove that is impossible to write $\sqrt{2} = \frac{m}{n}$ where m, n are integers
- Let x be a real number and $0 < x < 1$. Show that if n is the smallest positive integer such that $x - 1/n \geq 0$, then $x - 1/n < 1/n$

(Optional) Proof by contrapositive

- The result provide the grounding for a method of proof called **proof by contrapositive**
- In short, because $p \to q$ is equivalent to $\neg q \to \neg p$, we can prove a statement by it's contrapositive, which is sometimes easier
- Consider the following claim:
- **Claim:** Let $x \in \mathbb{Z}^+$. If $x^2 - 6x + 5$ is even, then x is odd.
- How can we prove this claim?

(Optional) Proof by contrapositive

- Instead proving directly, we prove it's contrapositive.
- The contrapositive of the statement is:
- **Contrapositive Claim:** If x is even, then $x^2 - 6x + 5$ is odd
- This is almost trivial to prove, because:

$$x^2 - 6x + 5 = \overbrace{x(x-6)}^{\text{even if x even}} + \overbrace{5}^{\text{odd}}$$

So the sum must be odd

Boolean arithmetic

- We have just looked at conditional statements that are connected using not, or, and operators
- In fact there are algebraic structure associated with these operators called **boolean arithmetic**
- Understanding these arithmetic rules can help us simply and rewrite our conditional statements

Properties of boolean operators

- Associativity
  1. $x \vee (y \vee z) = (x \vee y) \vee z$
  2. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
- Commutativity
  1. $x \vee y = y \vee x$
  2. $x \wedge y = y \wedge x$
- Distributive of $\wedge$ over $\vee$
  1. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
  2. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Properties of boolean operations

- Identities
  1. $x \vee F = x$
  2. $x \wedge T = x$
- Annihilators
  1. $x \vee T = T$
  2. $x \wedge F = F$
- Double negation
  1. $\neg(\neg x) = x$
- De Morgan's law
  1. $\neg(x \vee y) = \neg x \wedge \neg y$
  2. $\neg(x \wedge y) = \neg x \vee \neg y$

Exercise

Prove the following Identities:

1. $x \wedge (x \vee y) = x$
2. $\neg x \wedge y = \neg(\neg y) \wedge \neg x$
3. $a = b \wedge a$ if and only if $b = a \vee b$
4. Prove by the De Morgan's law by truth table
5. Prove by the distributive laws by truth table
6. Define $x \rightarrow y = \neg x \wedge y$. Show that $(x \wedge y) \rightarrow y = T$
7. Define $x \leftrightarrow y = (x \rightarrow y) \wedge (y \rightarrow x)$. Show that $x \leftrightarrow y = (x \wedge y) \vee (\neg x \wedge \neg y)$
8. Simplify$(x \vee y) \wedge \neg(\neg x \wedge y)$