CS1010 Tutorial 3

Agenda for Today

- Problem Set 8
- Problem Set 9
- Problem Set 10
- Assignment 1 (Practical)

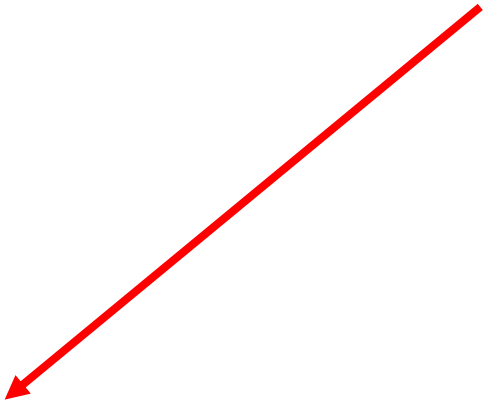
Problem 8.1

- Given two programs that find the **factorial** of a number

- Do they behave the same way?

Problem 8.1 – What is the behaviour?

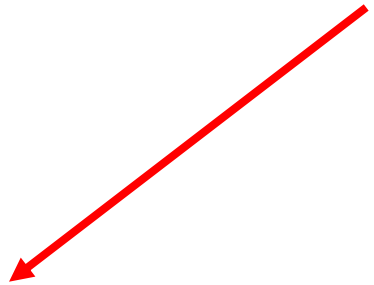
```
long factorial(long n)
{
    long answer;
    if (n == 0) {
        answer = 1;
    }
    answer = n * factorial(n - 1);
    return answer;
}
```



Recursive call always happens regardless of value of **n**

Problem 8.1 – What is the behaviour?

```
long factorial(long n)
{
    long answer;
    if (n == 0) {
        answer = 1;
    } else {
        answer = n * factorial(n - 1);
    }
    return answer;
}
```



Recursive call only happens when *n != 0*.

If initial **n** is greater than **0**, recursion always terminates

Problem 8.2

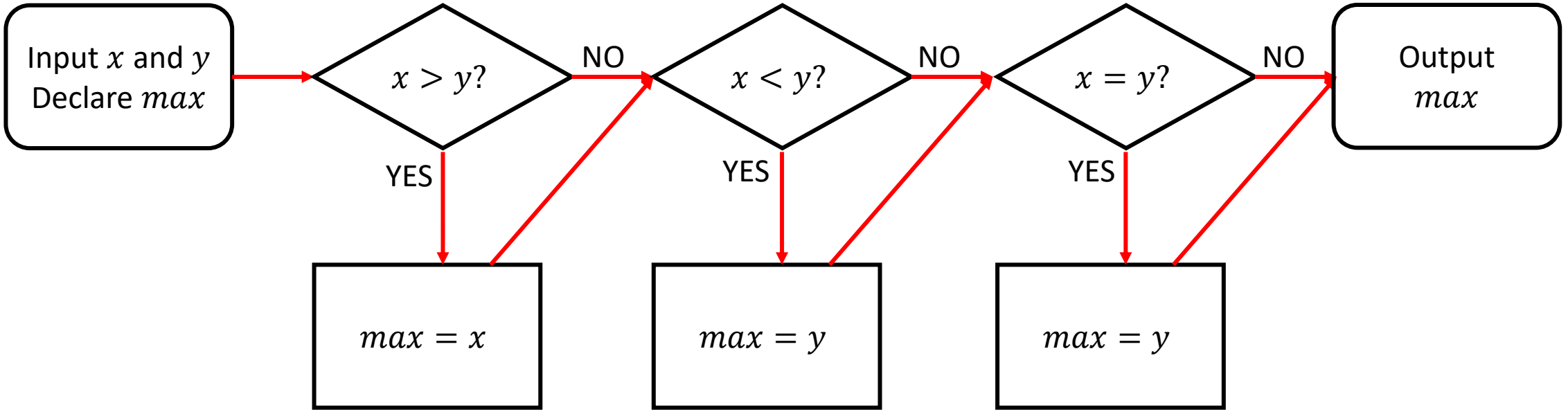
- Given three code snippets to **find the max of two numbers**

- Draw the **flowchart** for each part

Problem 8.2(a)

```
if (x > y) {
    max = x;
}
if (x < y) {
    max = y;
}
if (x == y) {
    max = y;
}
```

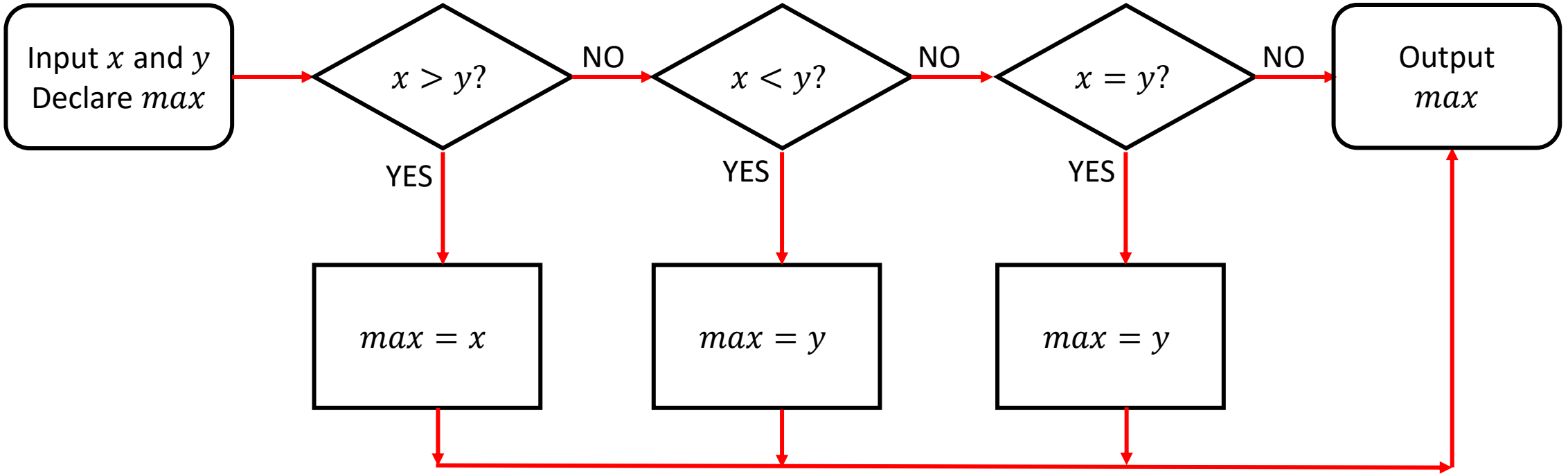
Problem 8.2(a)


Input x and y
Declare max
$x > y$?
NO
$x < y$?
NO
$x = y$?
NO
Output
max
YES
YES
YES
$max = x$
$max = y$
$max = y$

Problem 8.2(b)

```
if (x > y) {
    max = x;
} else if (x < y) {
    max = y;
} else if (x == y) {
    max = y;
}
```

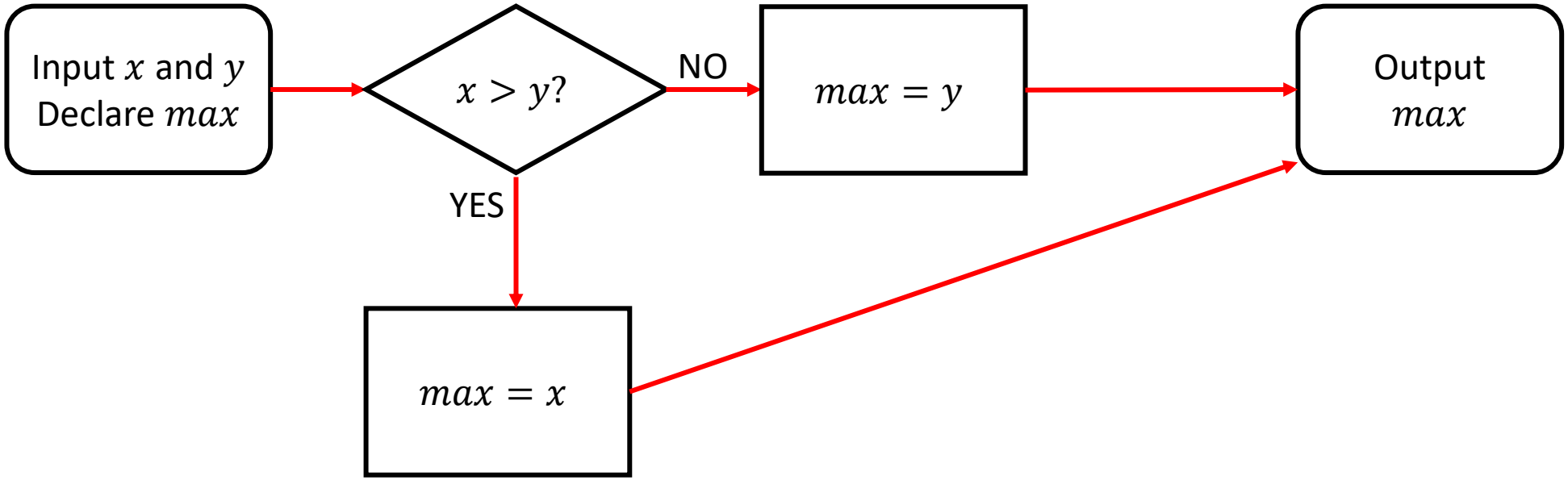
Problem 8.2(b)


Input x and y
Declare max
$x > y$?
NO
$x < y$?
NO
$x = y$?
NO
Output
max
YES
YES
YES
$max = x$
$max = y$
$max = y$

Problem 8.2(c)

```
if (x > y) {
    max = x;
} else {
    max = y;
}
```

Problem 8.2(c)



Avoid Redundant Checks

- **All code snippets find the max correctly**

- But (c) is the **most efficient** way to determine the max of two variables

- Why?
  - (a) checks **3** conditions regardless of the value of x or y
  - (b) checks for $x = y$ even if at the third **else**, it's guaranteed to be true
  - (c) only performs a single check

Problem 8.3 – Write as if...else statements

Score	Letter Grade
5 or higher	See Table 3
Less than 5	See Table 4

Score	Letter Grade
8 or higher	A
Less than 8	B

Score	Letter Grade
3 or higher	C
Less than 3	D

Problem 8.3

```
if (n >= 5) {
    // Table 3
    if (n >= 8) {
        return A;
    } else {
        return B;
    }
} else {
    // Table 4
    if (n >= 3) {
        return C;
    } else {
        return D;
    }
}
```

Problem 9.1

- Given two Boolean variables a and b, write out the truth conditions of the following expressions
  - (Do on board)

a	b	a && b	a || b	! a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Short-circuiting

- Notice the table for **AND** and **OR**

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Short-circuiting

- Notice the table for **AND** and **OR**

a	b	a && b
true	true	true
true	false	false
false	-	false
false	-	false

a	b	a || b
true	-	true
true	-	true
false	true	true
false	false	false

"-" denotes 'don't care' values

Short-circuiting

- Why is this important? Take a look at the following code:

```
bool is_positive_quotient(long a, long b)
{
    if (b != 0 && (a / b >= 0)) {
        return true;
    }
    return false;
}
```

Never evaluated.
Because `b != 0` is false
C knows not to check the other expression

Short-circuiting

- What happens if we change it to…

```
bool is_positive_quotient(long a, long b)
{
    if ((a / b >= 0) && b != 0) {
        return true;
    }
    return false;
}
```

Floating point exception
(core dumped)

Problem 9.2

a. What is wrong with the code?

b. Give a value a, b and c that exposes the bug

c. Fix the code to remove the bug

d. Replace the three **if** statements with **if…else** statements. Draw the flowchart

```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if ((a > b) && (a > c)) {
        // a is larger than b and c
        max = a;
    }
    if ((b > a) && (b > c)) {
        // b is larger than a and c
        max = b;
    }
    if ((c > a) && (c > b)) {
        // c is larger than a and b
        max = c;
    }
    return max;
}
```

Problem 9.2(a)

What is wrong with the code?

- The code looks OK at first glance
- What if $a = b = c$?
  - In this case, none of the **if** conditionals pass

```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if ((a > b) && (a > c)) {
        // a is larger than b and c
        max = a;
    }
    if ((b > a) && (b > c)) {
        // b is larger than a and c
        max = b;
    }
    if ((c > a) && (c > b)) {
        // c is larger than a and b
        max = c;
    }
    return max;
}
```

Problem 9.2(b)

Give a value a, b and c that exposes the bug

- Let $a = b = c \neq 0$

- Are there other possible alternatives?
  - What if $a = 2, b = 2, c = 1$?
    - Does not work
  - What if $a = 2, b = 1, b = 1$?
    - Works!

```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if ((a > b) && (a > c)) {
        // a is larger than b and c
        max = a;
    }
    if ((b > a) && (b > c)) {
        // b is larger than a and c
        max = b;
    }
    if ((c > a) && (c > b)) {
        // c is larger than a and b
        max = c;
    }
    return max;
}
```

Problem 9.2(b)

Give a value a, b and c that exposes the bug

- What are the possible relationships between a, b and c?
  - $a = b = c$
  - $a > b = c$
  - $a < b = c$
  - $a = b < c$
  - $a = b > c$
  - $a < c = b$
  - $a > c = b$
  - …

```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if ((a > b) && (a > c)) {
        // a is larger than b and c
        max = a;
    }
    if ((b > a) && (b > c)) {
        // b is larger than a and c
        max = b;
    }
    if ((c > a) && (c > b)) {
        // c is larger than a and b
        max = c;
    }
    return max;
}
```

Problem 9.2(c)

Fix the code to remove the bug

- The easiest way would be to change all $>$ to $\geq$.

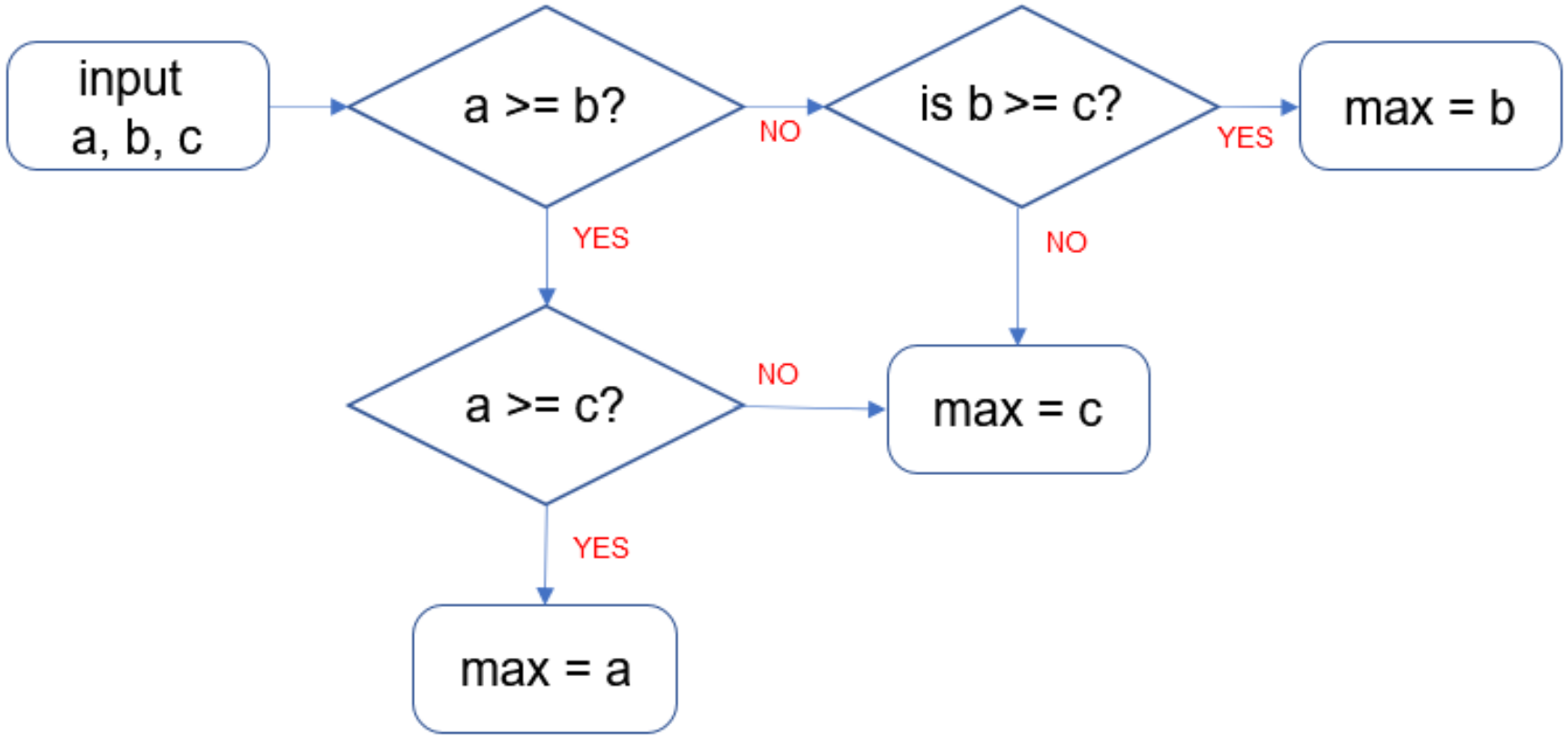
```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if ((a > b) && (a > c)) {
        // a is larger than b and c
        max = a;
    }
    if ((b > a) && (b > c)) {
        // b is larger than a and c
        max = b;
    }
    if ((c > a) && (c > b)) {
        // c is larger than a and b
        max = c;
    }
    return max;
}
```

Problem 9.2(d)

Replace the three if statements with if…else statements. Draw the flowchart

```
long max_of_three(long a, long b, long c)
{
    long max = 0;
    if (a >= b) {
        if (a >= c) {
            max = a;
        } else {
            max = c;
        }
    } else if (b >= c) {
        max = b;
    } else {
        max = c;
    }
    return max;
}
```

Problem 9.2(d)



Problem 9.2

- How I would do this:
- **By not chaining if…else**
- Put a, b, c into a List, then apply $FIND\_MAX$ on the List

Problem 9.3

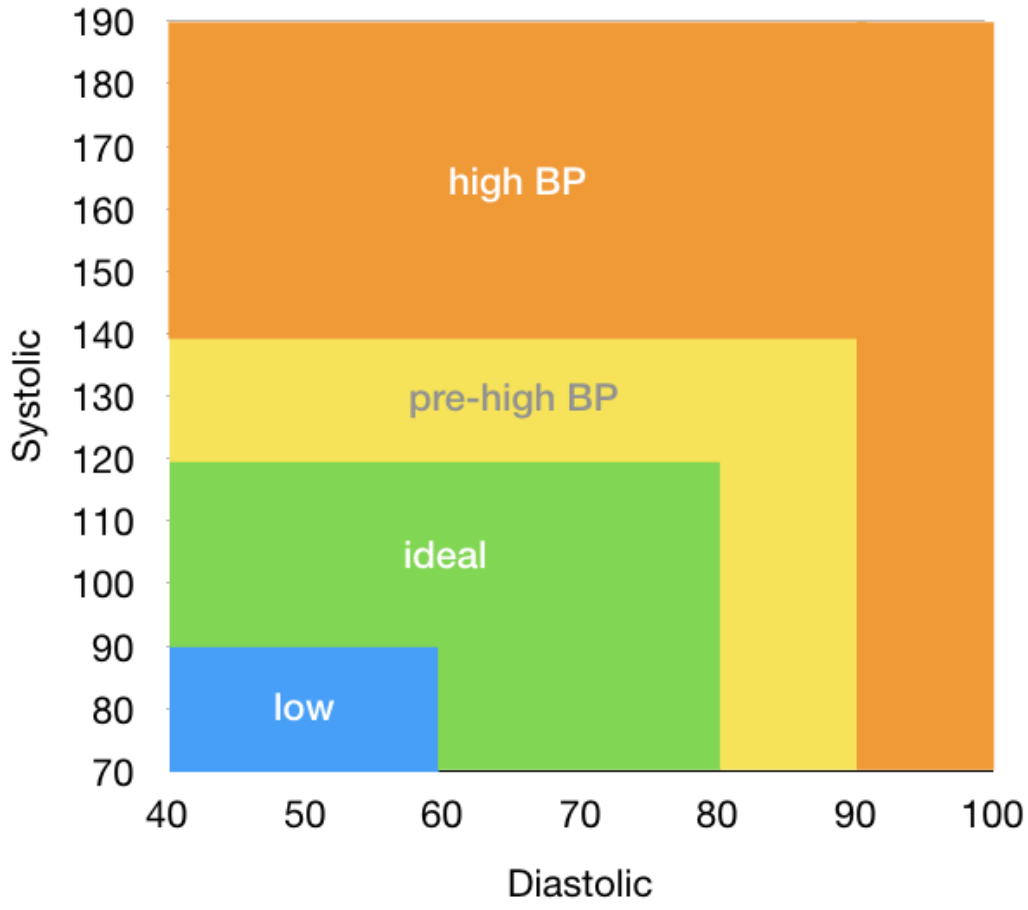
```
void print_blood_pressure(long systolic,
        long diastolic)
{
    if systolic <= 90 && diastolic <= 60
        print "low“

    else if systolic <= 120 && diastolic <= 80
        print "ideal“

    else if systolic <= 140 && diastolic <= 90
        print "pre-high bp“

    else
        print "high bp"

}
```



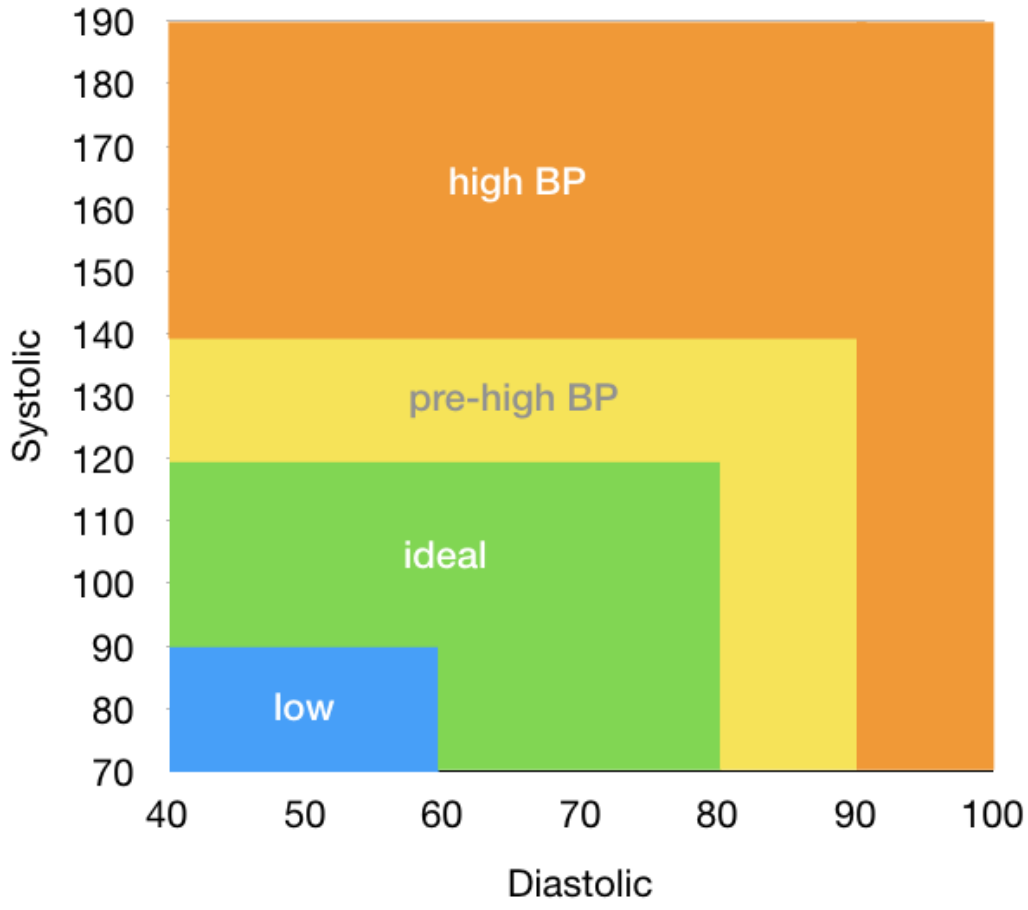
Problem 9.3

```
void print_blood_pressure(long systolic,
        long diastolic)
{
    if systolic > 140 || diastolic > 90
        print “high”

    else if systolic > 120 || diastolic > 80 {
        print “pre-high bp”

    else if systolic > 90 || diastolic > 60
        print “ideal”

    else
        print “low”
}
```



Problem 10.1

- Negate the following logical expressions
- Then apply De Morgan's Law to simplify the resulting expression.
- What is De Morgan’s Law?
  - `!(a && b) === !a || !b`
  - `!(a || b) === !a && !b`

Problem 10.1

- **(x > 1) && (y != 0)**
  - Negate: `!((x > 1) && (y != 0))`
  - Apply DML: `!(x > 1) || !(y != 0)`
  - Simplify: `x <= 1 || y == 0`

- **!eating && drinking**
  - Negate: `!(!eating && drinking)`
  - Apply DML: `eating || !drinking`

- **(has_cs2030 || has_cs2113) && has_cs2040c**
  - Negate: `!((has_cs2030 || has_cs2113) && has_cs2040c)`
  - Apply DML: `!(has_cs2030 || has_cs2113) || !has_cs2040c`
  - Apply DML: `(!has_cs2030 && !has_cs2113) || !has_cs2040c`

Problem 10.2

```
long score = 4;
if (something) {
  score = 10;
} else {
  score = 0;
}
// { ??? } { score == 10 || score == 0 }

if (score == 4) {
    score = 1;
} else {
    score += 10;
}
// { ??? } { score == 20 || score == 10 }

if (score >= 10) {
    cs1010_println_string("ok");
} else {
    cs1010_println_string("failed");
}
```

score != 4 no matter what, so “else” block always taken

score >= 0, so “ok” always printed

Assignment 1

Box, Digits, Suffix, Taxi

General Format of Assignments

- Between 1-4 coding problems
  - Write a program to solve each problem

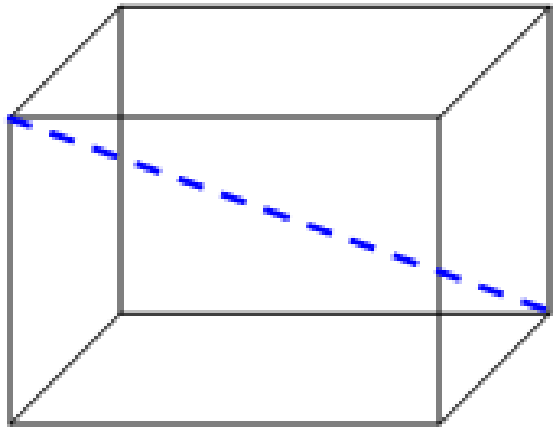
- Each problem reads input from **standard input**, and prints to **standard output**

Learning Outcomes

- Be able to write simple C programs that involve
  - Arithmetic operations
  - Conditional Branches (if…else)
- Familiarity with C datatypes
  - Long (integer)
  - Double (floating point)
  - Bool (true/false)
- **Breaking down a large problem into smaller sub-problems**
  - Writing a **function** to solve each sub-problem
  - Combine these functions to solve the overall problem

Question 1: Box – Overview

- Task – find the **diagonal** of a box



Dotted blue line shows the diagonal connecting two vertices that are furthest apart.

Question 1: Box – What is needed?

- Simple exercise in C datatypes and functions

- Write a new function called **area_of_rectangle** that computes the area of a rectangle
  - What parameters should it have?
    - Width of rectangle
    - Length of rectangle
  - What type should its parameters have? What type should it return?
    - Integer (long) or Decimal (double)?

Question 1: Box – What is needed?

- Modify the function given in the lecture – **hypotenuse_of**
  - What should you modify?
  - Hint given in the question

- Recommended: write two new functions that each
  1. Compute the surface area of the box
  2. Compute the diagonal of the box using **hypotenuse_of**

Question 2: Digits and Ordinal Suffix

- Digits – tests on **linear recursion** in code
- Suffix – tests on **if…else** chain

- How do you get the last (or first) digit of a number?
  - `long n = d % 10` sets **n** to the last digit of **d**

- How do you "strip" away the last (or first) digit of a number?
  - `d /= 10` or `d = d / 10`
  - We are using integer division to truncate the decimal part for us

Question 2: Digits – Tips

- One extremely big hint:
- Look at the **factorial(n)** function discussed in lecture
- The recursion style are largely the same

Question 3: Ordinal Suffix – Tips

- Consider what cases you need
- Which are the strictest cases?

Question 4: Taxi

- The problem looks tricky, but is actually quite easy

- Break down the problem (the question tells you how)
  - How to find the surcharge?
    - What cases of the surcharge are there?
  - How to find the fare?
    - What cases of the fare are there?

- Write a function to solve each problem, and combine them to solve the overall problem

Question 4: Taxi – How to find the surcharge?

Surcharge		
Monday to Friday	6:00 to 9:29	25% of metered fare
Daily	18:00 to 23:59	25% of metered fare
Daily	0:00 (midnight) to 5:59	50% of metered fare

Question 4: Taxi – How to find the surcharge?

- How do we calculate the surcharge? ✓

- Sub-questions
  - How do we determine **what day of the week** is it? ✓
  - Given the time
    - Is it 0600 – 0929? ✓
    - Is it 1800 – 2359? ✓
    - Is it 0000 – 0559? ✓
  - How do you check each of this
  - **Use as few parameters needed, try not to have functions that have unnecessary parameters**

- Each ✓ is a possible function

Question 4: Taxi – How to find the base fare?

Basic Fare	
The first 1 km or less (Flag Down)	$3.20
Every 400 m thereafter or less, up to 10 km	$0.22
Every 350 m thereafter or less, after 10 km	$0.22

Question 4: Taxi – How to find the base fare?

- What cases are there for the base fare given the distance d?
  - $d \leq 1000$ ✓
  - $1000 < d \leq 10000$ ✓
  - $d > 10000$ ✓

- All information given in the table

- The **ceil()** function in the **math.h** header might be useful

- Remember to **check your datatypes**