FAST NUCES Programming Olympics

PROGRAMMING FUNDAMENTAL CS1002 - Fall 2024

Problem 1: Miles per hour to kilometers per seconds

This year olympics were held in Europe causing all sorts of problems for the Americans as they don't use the standard measures like meters and kilometers, grams and kilograms. For racing events the speeds are measured in kilometers per seconds. Can you design a solution to convert kilometers per seconds to miles per hour for US viewers of the Paris Olympics?

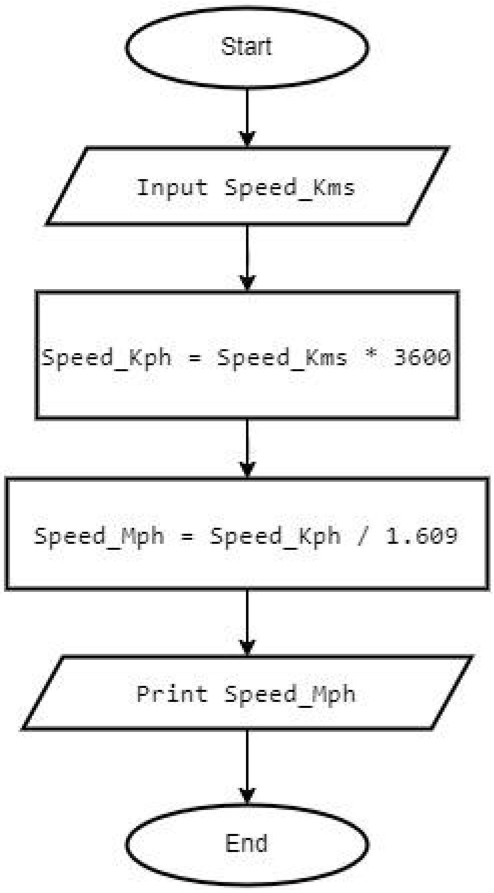
Solution:

This is a warm-up question and isn't much of a challenge. The student must exhibit the correct use of input and output and realize that both of them are single numerical values. For processing the conversion formula must be mentioned. The flowchart and pseudocode are trivial and straightforward with a single flow without any branching or loops.

Pseudocode:

```
01.Start
02.Input Speed_Kms
03.Speed_Kph = Speed_Kms * 3600
04.Speed_Mph = Speed_Kph / 1.609
05.Print Speed_Mph
06.End
```

Flowchart:



Problem 2: Even or Odd

Mr. Bhola does not know what an even or odd number is. He is least bothered to learn the difference either. It all comes down to your programming skills to design a program that will ask for a positive number greater than 0 and prints whether the number is even or odd. Be careful, Mr. Bhola can even enter 0 or negative numbers so your solution must give an error saying invalid input in that case.

Solution:

IPO is trivial with input being a single integer greater than 0 and output either being Odd or Even. The solution includes checking for even number using the mathematical definition N%2 == 0 or N/2 * 2 == N where N/2 must be integer division. Other solutions might also exist. The flowchart must depict one branching and equivalent pseudocode must check for at least one condition using the if statement.

Pseudocode:

```
01.Start
02.Input number
03.If number <= 0
04.      Print “Invalid Input”
05.Else
06.If number MOD 2 == 0
07.      Print “Even”
08.Else
09.      Print “Odd”
10.End
```

Problem 3: Legal age of marriage in Pakistan

Child marriages are a major problem in Pakistan. To avoid responsibility the Federal Government delegated the task of Child Marriage Laws to provinces. Only Punjab and Sindh have updated the laws while in Punjab the law is going under another amendment this year. To make things simple in Sindh, No child, girl or boy, below the age of 18 can marry. Whereas in Punjab (until the new bill is passed) and other provinces ACT of 1929 is followed which prohibits girls below the age of 16 to marry whereas for boys the age is 18.

You are required to design a solution for both provinces to make this decision easy by asking relevant information from the user and giving output whether it is legal to marry to or not.

Solution:

Input:

Province (Punjab, Sindh, or others)

Age of the groom (in years)

Age of the bride (in years)

Process:

Determine Province Specific Law:

Sindh: Age of both groom and bride must be 18+.

Punjab: Until new bill is passed (implementation TBD), the 1929 Act is followed:

Bride's age must be 16+ and Groom's age must be 18+.

Other Provinces: Follow 1929 Act:

Bride's age must be 16+ and Groom's age must be 18+

Check Legal Marriage Age:

Assess if the user provided ages fit within the determined provincial legal age limits.

Determine the Output:

Legal to Marry: When individual ages are within legal limitations set by law of specified regions

Not Legal to Marry: When individual ages breach the legal limitations according to the specified region's law.

Output:

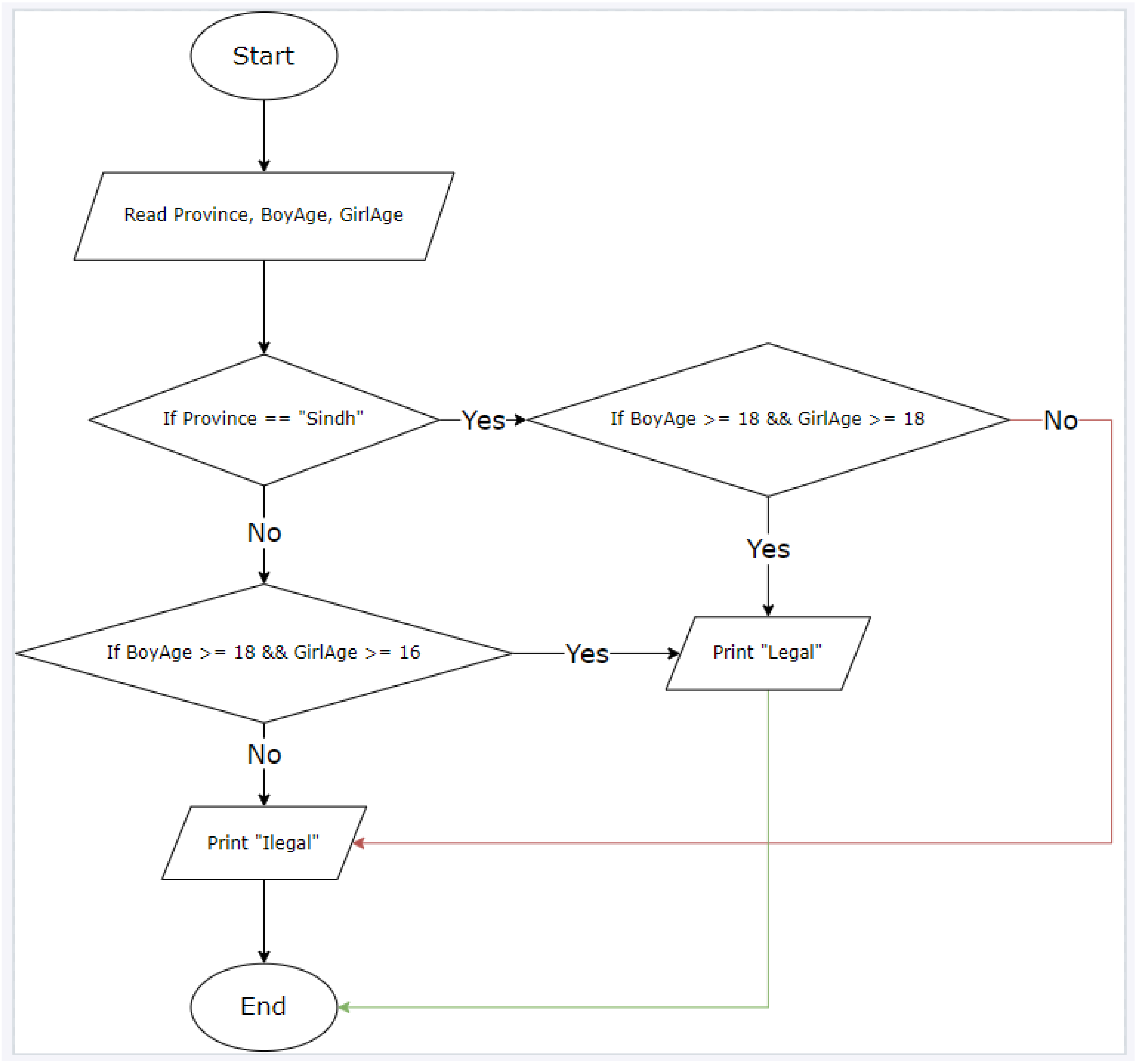
An informative and clear statement: "The marriage is legal/"illegal considering your given ages."

Flowchart and Pseudocode:

Many flowcharts and pseudocodes are possible. The flowchart solution must demonstrate correct usage of the decision boxes and whether the result is correct on all possible inputs. Same holds true for pseudocode and focus should be on correct logic rather than one specific pseudocode.

Alternate understanding:

It can also be interpreted that from the language of the question that one only has to clarify whether an individual (Girl or Boy) qualifies to marry or not instead of considering both bride and groom.



Start
Read Province, BoyAge, GirlAge
If Province == "Sindh"
Yes
If BoyAge >= 18 && GirlAge >= 18
No
No
Yes
If BoyAge >= 18 && GirlAge >= 16
Yes
Print "Legal"
No
Print "Ilegal"
End

Problem 4: Grocery Assistance/ Calculator

Imagine Mr. Bhoola is going to the market to buy some yummy fruits and vegetables. He's a little bit forgetful, so he doesn't always have a list or a fixed amount of money to spend. We're going to help Mr. Bhoola figure out how much he needs to pay and how much change he gets back.
Here's what we need to know:

- How much money Mr. Bhoola has today to buy fruits and vegetables.
- The prices of the onions, apricots, grapes, and tomatoes at the market.
- What vegetables Mr. Bhoola buys and how many of each.

Design a solution to help Mr. Bhoola every time he goes shopping!

IPO Chart:

Input	Process	Output
- Total available money ($)	- Calculate the total cost of each type of vegetable	- Total cost of the purchase ($)
- Vegetable name	- Multiply cost per kg by the number of kg for each vegetable	- Amount of change ($)
- Vegetable quantity (kg)	- Sum of the individual vegetable costs	
- Cost of each vegetable per kg	- Compare total cost of the purchase with the available money	
	- Calculate the change (if any)	

Flowchart & Pseudocode

Many different solutions to designing a flowchart exist. Below is a pseudocode of one such flowchart.

```
1.  Start
2.  Input Quantity_Onions, Quantity_Apricots, Quantity_Grapes, Quantity_Tomatoes
3.  Input Price_Onions, Price_Apricots, Price_Grapes, Price_Tomatoes
4.  Input Total_Money
5.  Total_Cost = Quantity_Onions * Price_Onions   +
                  Quantity_ Apricots * Price_ Apricots   +
                  Quantity_ Grapes * Price_ Grapes   +
                  Quantity_ Tomatoes * Price_ Tomatoes
6.  If Total_Cost > Total_Money
7.       Print “Not Enough Money”
8.  Else if Total Cost  ==  Total_Money
9.       Print “No Change”
10. Else
11.      Print  Total_Money  –  Total_Cost
12. End
```

Problem 5 : Crop Management.

A smart agricultural drone monitors a large farm to manage irrigation efficiently. The drone first checks the type of crop being monitored whether it's wheat, corn, or rice. For wheat, if the soil moisture is below 30%, and if it has not rained in the last 24 hours, the system will initiate irrigation. For corn, if the soil moisture is below 40% , the system will irrigate regardless of recent rainfall. For rice, if the soil moisture is below 25% , the system will irrigate unless it has rained in the last 24 hours. If the soil moisture is above these thresholds, no irrigation is applied. Design a solution to determine if irrigation should be initiated based on the crop type, soil moisture level, and rainfall status.

Solution Approach: We will develop a decision-making algorithm that uses logical comparisons to determine the necessity for irrigation.

IPO Chart:

Input	Process	Output
Crop Type (Wheat, Corn, Rice) Soil Moisture (%) Rainfall Status (True/False)	Check Soil Moisture Level, Check Rainfall Status, Apply Logic based on Crop Type Compare Soil Moisture to Threshold for respective crop	Irrigation On/Off

Flowchart and Pseudocode: Many possible flowcharts. Pseudocode for one such flowchart shown.

```
Start  irrigation_decision
    Input -> crop_type, soil_moisture,
    Input -> rained_in_last24 | true/false.
    if crop_type == "Wheat":
        if soil_moisture < 30 and not rained_in_last24:
            print("Irrigation initiated")
        else:
            print("No Irrigation required")
    else if crop_type == "Corn":
        if soil_moisture < 40:
           print("Irrigation initiated")
        else:
            print("No Irrigation required")
    else if crop_type == "Rice":
        if soil_moisture < 25 and not rained_in_last24:
            print("Irrigation initiated")
        else:
            print("No Irrigation required")
    else:
        print  ("Invalid Crop Type")
End
```

Problem 6: Amusement Park Ride eligibility

Safari park is a famous amusement park situated in Karachi that has specific height and age requirements for its rides. For The Dragon Roller Coaster, visitors must be at least 48 inches tall and 10 years old. The Sky Swing requires visitors to be at least 54 inches tall, with no age restriction. The Carousel has no height requirement but visitors must be at least 5 years old. A visitor is wearing a band so he may take as many rides as he wants until he is tired. Design a solution that determines if a visitor meets the criteria to go on a chosen ride based on their height, age, and selected ride.

Examples:

Input: ● Height: 55 inches ● Age: 10 years ● Ride: "The Carousel" **Expected Output:** "You meet the criteria for The Carousel."	**Input:** ● Height: 48 inches ● Age: 9 years ● Ride: "The Dragon Roller Coaster" **Expected Output:** "Sorry, you do not meet the criteria for The Dragon Roller Coaster."

Solution approach

We will design a decision-making algorithm that continuously checks whether a visitor meets the criteria for selected rides based on height, age, and ride requirements until the visitor decides they are tired.

IPO Chart

Input	Process	Output
Visitor's Height (inches) Visitor's Age Selected Ride Tired Status (True/False)	Check if the visitor meets the height requirement for the selected ride based on input Check if the visitor meets the age requirement for the selected ride Apply specific height and age criteria for the selected ride (Dragon Roller Coaster, Sky Swing, Carousel) Continuously check if the visitor is tired and wants to stop taking rides	Ride Access: Allowed/Denied

Pseudocode and Flowchart

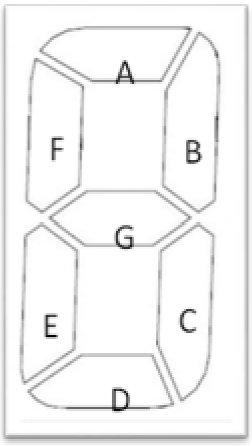
Many possible solutions. Pseudocode of one such solution is provided. Equivalent flowchart is trivial.

```
Start ride_access_decision
    Input -> height, age, tired (true/false)
    while tired == false:  # Loop until the visitor gets tired
        Input -> ride_selected  # Ask for the next ride selection
        if ride_selected == "Dragon Roller Coaster":
            if height >= 48 and age >= 10:
                print("Access to Dragon Roller Coaster allowed")
            else:
                print("Access to Dragon Roller Coaster denied"
        else if ride_selected == "Sky Swing":
            if height >= 54:
                print("Access to Sky Swing allowed")
            else:
                print("Access to Sky Swing denied")
        else if ride_selected == "Carousel":
            if age >= 5:
                print("Access to Carousel allowed")
            else:
                print("Access to Carousel denied")
        else:
            print("Invalid ride selected")
                # Check if the visitor is tired
        Input -> tired (true/false)  # Update tired status (e.g., visitor decides if
   they're tired)
            print("Visitor is tired, no more rides.")
End
```

Problem 7: Which floor…

IPO

Input	**Processing**	**Output**
Single 7 digit number Of only 1s and 0s	extract digits one by one using %10 to determine which floor	floor number



	ABCDEFG		ABCDEFG
1 =	0110000	2 =	1101101
3 =	1111001	4 =	0110011
5 =	1011011	6 =	1011111
7 =	1110000	8 =	1111111
9 =	1111011	0 =	1111110

Pseudocode and flowchart

Many possible solutions and flows can be taken. Below is one of the pseudocode. Flowchart is trivial.

```
Start:
  Input -> binary // a seven digit number of only 1s and 0s
  Let:
    var FG = binary % 100 //stores info of panels F and G
    var AtoE = binary / 100 //stores info of all panels except F and G
    var AB = binary / 100000
    if FG == 11 then //either its 4,5,6,8, 9
        if(AB == 11) //its 8 or 9
            if(binary % 1000 < 100) Write “Its Floor 9”
            else Write “Its Floor 8”
        else if(AB == 10) //either its 5 or 6
            if(AtoE % 10 == 0) Write “Its Floor 5”
            else Write “Its Floor 6”
        else if(AB == 1) Write “its floor 4”
    else if (FG == 10) //its 0
        Write “Its floor 0”
    else if (FG==1) //its either 2,3,
        if(binary % 1000 > 100) Write “Its floor 2”
        else Write “Its floor 3”
    else //only floor 7 is left
        Write “Its floor 7”
End
```

Problem 8: Digit Sum

You are required to design a solution that takes as input a single number and prints out the sum of its digits. Check the examples below for clarity.

Examples:

sumDigit(345) = 12
sumDigit(55) = 10
sumDigit(1) = 1
sumDigit(400000000) = 4
sumDigit(1600) = 7
sumDigit(0) = 0

Input	Process	Output
Single number	Sum the digits of the number.	Sum of the digits.

Pseudocode:

```
Start:
 Input from user into var N
 sum = 0
  while N > 0:
      digit =N % 10
      sum = sum + digit
      N = N / 10 //integer division
  End while
write “Sum of Digits is: “ + sum
End
```

Problem 9 : Exact Age in days, months, and years from DOB

Solution

Input	**Process**	**Output**
Birthdate (day, month, year) Currentdate (day,month,year)	Validate birthdate, calculate age based on current date.	Age (in years)

Pseudocode

```
Start
Take 6 number input from user bday, bmonth, byear, cday, cmonth, cyear
Validate these 6 inputs using switch and if statements
If month is 1 OR 3 OR 5 OR 7 OR 8 OR 10 OR 12 days are less than equal to 31
Else if month is 2
    Check for leap year and verify if days are less than 28 or 29 accordingly
Else
    Check if days are less than equal to 30.
# Calculate age in years
age_years = cyear - byear

# Calculate age in months
age_months = cmonth - bmonth
if age_months < 0:
  age_years-= age_years - 1
  age_months += 12

# Calculate age in days
if age_months == 0:
  age_days = cday - bday
else if current_day < birth_day:
  # Adjust for days in the previous month
  if bmonth == 2 and is_leap_year:
    max_days_in_previous_month = 29
  else:
    max_days_in_previous_month = 28 if bmonth == 2 else 31 - (bmonth % 2)
  age_days = cday + max_days_in_previous_month - bday
else:
  age_days = cday - bday
# Print the calculated age
print "Age:", age_years, "years", age_months, "months", age_days, "days"
End
```

Problem 10: Poor Mr. Bhoola bought a faulty keyboard

Imagine Mr. Bhoola's keyboard is a bit mischievous! Every time he types the number "9," the keyboard adds an extra zero. So, if he types "9," it shows up as "90".

Your Mission:

Mr. Bhoola wrote down some numbers using his silly keyboard. But we know they're wrong because of the extra zeroes. Your job is to fix them by taking away the extra zeroes whenever you see a "90" in the number.

Here are some examples:

100900: The keyboard typed "90" where there should be "9". So, it's really 10090.
1540090: The keyboard added an extra zero again. It's really 154009.
1290905: This one is tricky! There's a "90" where there should be a "9". It's really 12995.

Design a solution to fix the other numbers Mr. Bhoola wrote down!

Input	**Process**	**Output**
A number	Extract digits from the number.	Corrected number
	- Identify and remove "90" patterns, treating them as "9".	without extra zeros
	- Reconstruct the number without the "90" patterns.	

Solution Approach:

We need to scan the entire number, check for instances of "90," and replace them with "9" to correct the number.

Pseudocode

```
Start
    # Input: The wrong number
    Input -> wrong_number  # Example number, e.g., 1290905

    # Initialize variables:
    corrected_number = 0  # To store the final corrected number
    multiplier = 1  # This will help us build the number from right to left
    previous_digit = -1  # A flag to remember the last processed digit

    # Process the digits of the number from right to left:
    while wrong_number > 0:
        # Extract the last digit
        current_digit = wrong_number % 10

        # Remove the last digit from the number
        wrong_number = wrong_number // 10

        # Check if we are in a "90" pattern
        if current_digit == 0 and previous_digit == 9:
            # Skip this '0' as it represents an extra zero
            previous_digit = -1  # Reset previous_digit after detecting "90"
        else:
            # Add the current digit to the corrected number
            corrected_number = corrected_number + (current_digit * multiplier)
            # Update multiplier to place the next digit in the correct position
            multiplier = multiplier * 10
            # Update previous_digit
            previous_digit = current_digit

    # Output the corrected number
    print(corrected_number)  # Output the result

End
```

Problem 11: No not optimus prime its Coprime..

Mr. Bhoola just found out about coprime numbers. Now he is obsessed with them and keeps checking two numbers whether they are coprime or not. You are to design a solution to this problem and help Mr. Bhoola with checking whether any two whole numbers greater than 0 are coprime or not.

A simplified definition of Coprime numbers is that their GCD is 1. You are encouraged to check wiki or other resources for more information.

Examples:

8 and 9 are coprimes despite the fact that they are not prime numbers.

14 and 33 are coprimes.

Solution:

IPO

Input	**Process**	**Output**
Two whole numbers (greater than 0)	Check if the greatest common divisor (GCD) of the numbers is 1.	"Coprime" or "Not coprime"

Pseudocode:

```
Start:
print "Enter the first number (greater than 0):"
number1 = read_number()
print "Enter the second number (greater than 0):"
number2 = read_number()

smaller_number = number1 if number1 < number2 else number2 # Find the smaller number
divisor = 2 # Initialize a divisor to 2

# Keep checking divisors until the smaller number becomes 1
while divisor <= smaller_number:
  # If both numbers are divisible by the divisor, they are not coprime
  if number1 % divisor == 0 and number2 % divisor == 0:
    print "Not coprime"
    END

  # Increment the divisor
  divisor = divisor + 1

# If the loop completes without finding a common divisor, the numbers are coprime
print "Coprime"
END
```

Problem 12: Die Hard 3: Defuse the bomb… Quick!

You've got to defuse a bomb by placing exactly 4 liters of water on a sensor. And you have to be quick! The problem is, you only have a 5L jug and a 3L jug on hand!

See the video clip here: https://youtu.be/BVtQNK_ZUJg

You have an unlimited water source, and if needed you can also empty the water in the jugs to get rid of it.

How could 4 liters be measured?

Start measure_4_liters

Input	Process	Output
5-liter jug, 3-liter jug, unlimited water source	- Fill jugs with water. - Transfer water between jugs. - Empty jugs as needed. - Achieve exactly 4 liters in one of the jugs.	4 liters of water

```
Start:
 # Initialize jugs
    jug5 = 0  # Amount of water in the 5-liter jug
    jug3 = 0  # Amount of water in the 3-liter jug

    # Step 1: Fill the 5-liter jug
    jug5 = 5

    # Step 2: Pour water from the 5-liter jug into the 3-liter jug
    jug3 = 3  # 3-liter jug is now full
    jug5 = jug5 - jug3  # Remaining in 5-liter jug: 5 - 3 = 2

    # Step 3: Empty the 3-liter jug
    jug3 = 0

    # Step 4: Transfer the remaining water from the 5-liter jug to the 3-liter jug
    jug3 = jug5  # 3-liter jug now contains 2 liters of water
    jug5 = 0  # 5-liter jug is now empty

    # Step 5: Fill the 5-liter jug again
    jug5 = 5

    # Step 6: Pour water from the 5-liter jug into the 3-liter jug (which already has 2 liters)
    jug3 = jug3 + jug5  # 3-liter jug will be full (3 liters), 2 liters + 5 liters = 7 liters
    jug5 = jug5 - (3 - jug3)  # Remaining in 5-liter jug: 5 - 1 = 4

    # Step 7: Now, jug5 contains exactly 4 liters
    # Output the result
    print("Jug5 contains 4 liters of water")

End
```

Problem 13: The general N-M size die hard jug problem

Design a solution to solve the general form M-NL jugs of the above problem.

So instead of having just 3-5L jugs you can be given 3-9L jugs or 7-9L jugs. Can you extract any number of liters of water less than the larger jug or only some quantities. Is there a formula for this? What amounts of water can be extracted from M-NL jugs?

Input	Process	Output
Capacity of Jug 1 (M) Capacity of Jug 2 (N) Desired Amount of Water (D)	Check if the desired amount can be measured using the GCD rule. Manage water quantities based on capacities, ensuring no overflow or underflow.	Whether the desired amount can be measured or not.

```
Star

    # Input: Two jugs and desired water amount
    Input -> M  # Capacity of first jug (e.g., 9 liters)
    Input -> N  # Capacity of second jug (e.g., 3 liters)
    Input -> D  # Desired amount of water (e.g., 6 liters)

    # Initialize water quantities in both jugs to 0
    jug1_water = 0
    jug2_water = 0

    # Step 1: Check if D is greater than the larger jug capacity
    If D > M and D > N
        Print "Cannot measure the desired amount"
        Stop

    # Step 2: Check if D is exactly one of the jug capacities
    If D == M or D == N
        Print "Desired amount can be measured"
        Stop

    # Step 3: Calculate GCD without using functions
    # Ensure M is the larger value
    If M < N
        temp = M
        M = N
        N = temp

    # Calculate GCD by subtraction method
    original_M = M
    original_N = N
    While M != N
        If M > N
            M = M - N
        Else
            N = N - M

    # M (or N) now contains the GCD
    gcd_value = M

    # Step 4: Check if D can be measured using GCD logic
    # D must be a multiple of the GCD of the jug capacities
    If D % gcd_value == 0
        Print "Desired amount can be measured"
    Else
        Print "Desired amount cannot be measured"

End
```

GOOD LUCK 🙂