**CECS 174 - Project 2**

**“Water Utility Company”**

**Due date: 10/08/2022**

Team: team\_FoojanLesly

Student Name: Fozhan Babaeiyan Ghamsari

Student ID: 029701865

I certify that this submission is my original work

Fozhan Babaeiyan Ghamsari

<< Add one for each team member, submit one report as a team >>

Student Name: Lesly Castellanos Ibanez

Student ID: 027961126

I certify that this submission is my original work

Lesly Castellanos Ibanez

**Project Report: Programming Project 2 - “Water Utility Company”**

1. **Goal:** The goal of this project is to create a program for a water utility company that displays the customer’s code, beginning meter reading, ending meter reading, the gallons of water used, and the amount billed.
2. **Problem Description:** Given restricted values that the user may input, the program has to compute information based on the letter that the user inputs. These restricted values include ‘r’, ‘c’, and ‘i’. After receiving the letter, the user would be told to input the beginning meter reading and the ending meter reading. The program should work when the ending is less than the beginning by using the absolute value function and should also run when the character entered is not ‘r’, ‘c’, or ‘i’. Keeping this in mind, each letter has its certain ways of calculations/formulas. At the end, the output should be aligned, the amount billed is rounded to the nearest hundredth, and the gallons is rounded to the nearest tenth.
3. **Program Description:** 
   1. **Solution of the problem**: The various inputs that the user may enter are positive integers, negative integers, integers that have less than 9 digits, and characters that are out of the range of r,c,i. For the code ‘r’, the calculations include $5.00 plus the $0.0005 per gallon used. For code ‘c’ it is $1000 for 4 million gallons or less, and .00025 for each additional gallon used. For code ‘i’, it is: $1000 if the usage does not exceed 4 million gallons. $2000 if the usage exceeds 4 million gallons but does not exceed 10 million gallons. Lastly $2000 plus $0.00025 for each additional gallon if usage exceeds 10 million gallons. For the inputs that are negative, the math module would be used to convert them into positive values using the absolute value function math.fabs(). Additionally, for inputs that are less than 9 digits, the format function is used to format it to have 9 digits. However, what happens when the user enters an input that is out of the desired range? The code would be programmed to print out 0.0 for the gallons of water used and the amount billed as $0.00.
   2. **Test cases**: In order to check that our code works for all types of input scenarios, we imputed a series of different characters, positive, negative, and numbers that are less than 9 digits. We chose these test cases because they are the possible inputs that may be imputed in my program. Our code may be correct for certain inputs, but it has to be correct for all instances.

* inputs ⇒ outputs

-negative number ⇒ +positive number

z ⇒ 0.0 for gallons and $0.00 for bill

+positive number ⇒ +positive number

5 ⇒ 000000005

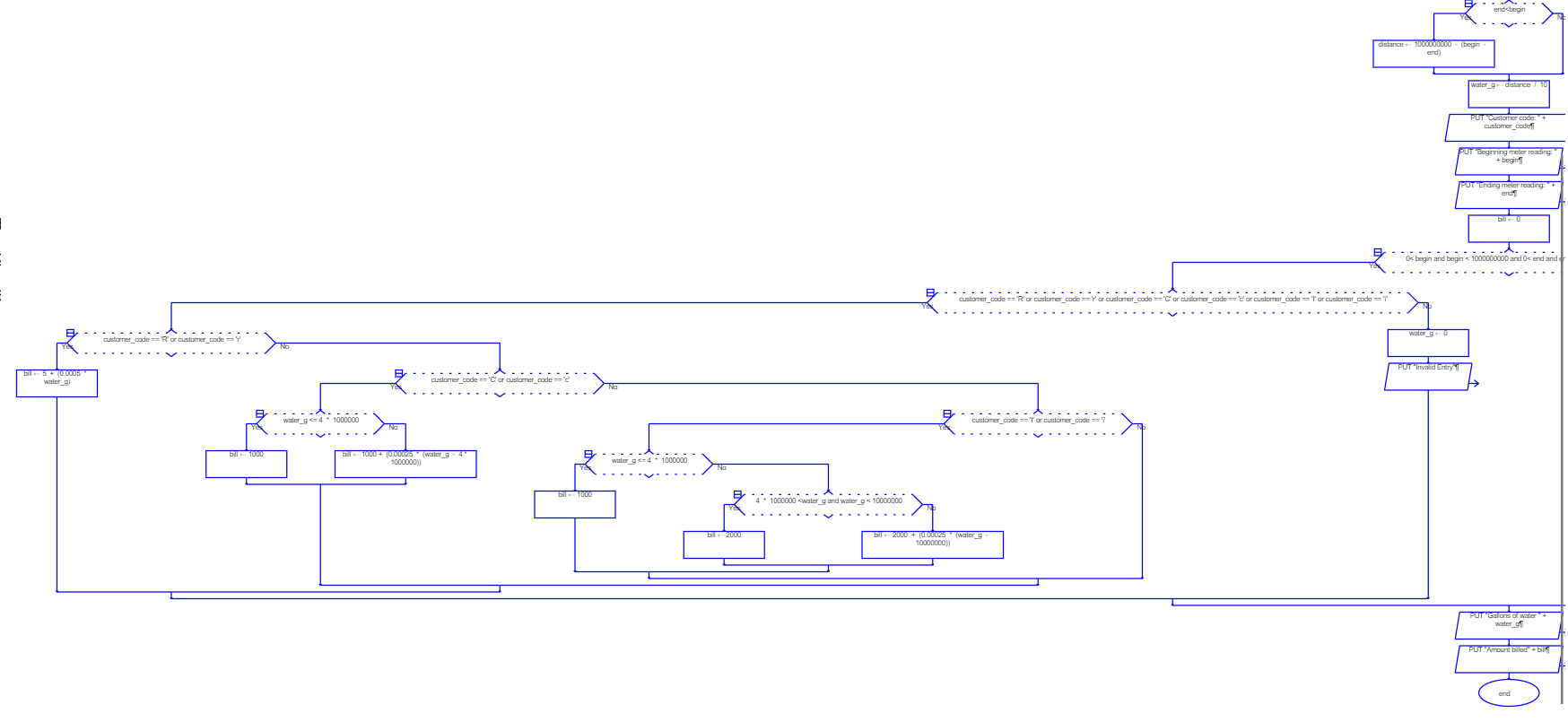
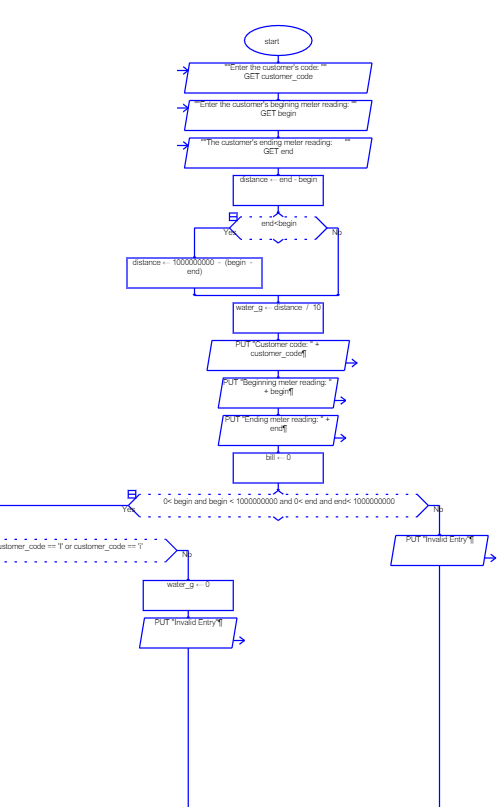
* 1. **Pseudocode:**
* **start**
* Import the math module
* Set variable customer code to ask the user to enter their customer code
* Set variable begin to ask the user to enter the beginning meter reading
* Set variable end to ask the user to enter their ending meter reading
* Set variable distance to be the end subtracted from begin
* Continue if the ending is less than the beginning
* If it is true, make the distance to be 10e8 subtracted by the absolute value of the ending minus the beginning.
* Set variable water gallons to be the distance divided by 10.
* Display the “customer code: “
* Display the “Beginning meter reading: “ and format it to have 9 digits in total
* Display the “Ending meter reading: “ and format it to have 9 digits in total while also being aligned with the beginning meter reading.
* Set variable bill to be 0
* Continue if variable begin is greater than 0 but is less than 10e8 and if end is greater than 0 but less than 10e8
* If it is true, continue if variable customer code is equal to ‘R’ or ‘C’ or ‘I’ or ‘r’ or ‘c’ or ‘i’
* If it is true, continue if the customer code is equal to ‘R’ or ‘r’
* If it is true, set the bill to be 5 plus the water gallons times 0.0005.
* Else if the customer code is equal to ‘C’ or ‘c’ and if the water gallons is less than 4 times 10e5, set the bill to be 1000.
* If false, set the bill to be the gallons of water minus 4, times 10e5, times 0.00025, and plus 1000.
* Else if the customer code is equal to ‘I’ or ‘i’ and if the water gallons is less than 4 times 10e5, set the bill to be 1000
* Else if the water gallons is greater than 4 times 10e5 but is less than 10e6, set the bill to be 2000
* If false, set the bill to be water gallons minus 10e6, times 0.00025, and plus 2000.
* If the customer code does not equal to ‘R’ or ‘C’ or ‘I’ or ‘r’ or ‘c’ or ‘i’, set the water gallons to be 0 and display “Invalid Entry”
* If the variable begin is not greater than 0 but less than 10e8 and if the ending is not greater than 0 or less than 10e8, then display “Invalid Entry”
* Display the “Gallons of water used: “ to be formatted to be in the tenths place
* Display the “Amount billed: “ to be formatted to be in the hundredths place
* **End**
  1. **IPO**

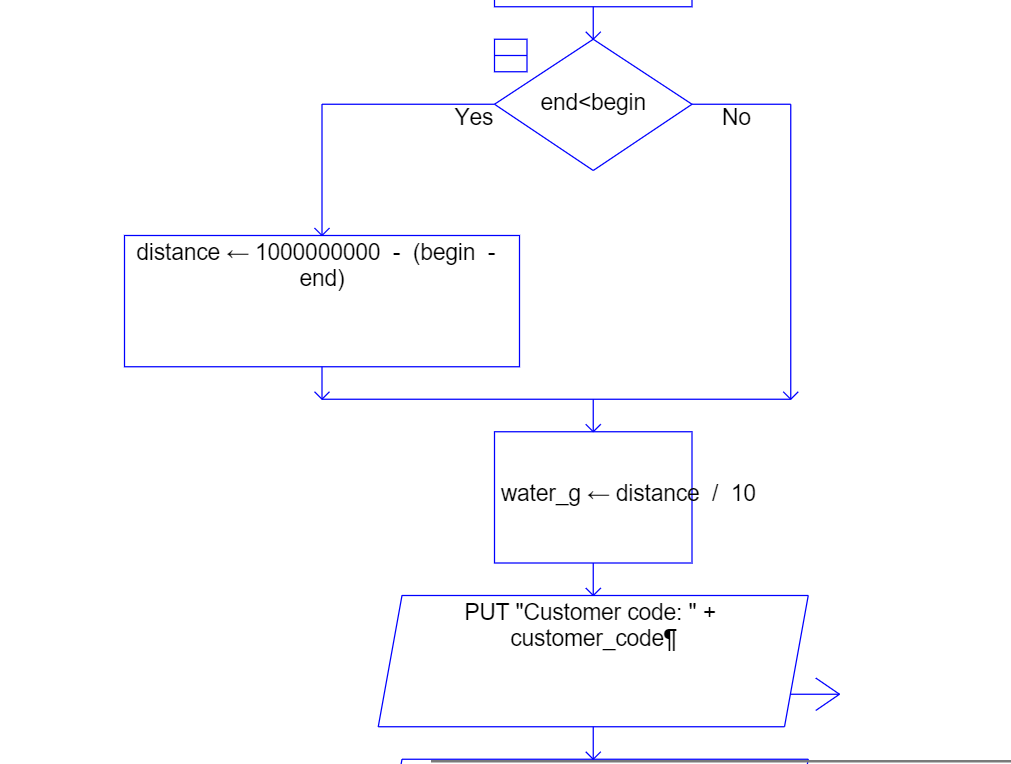
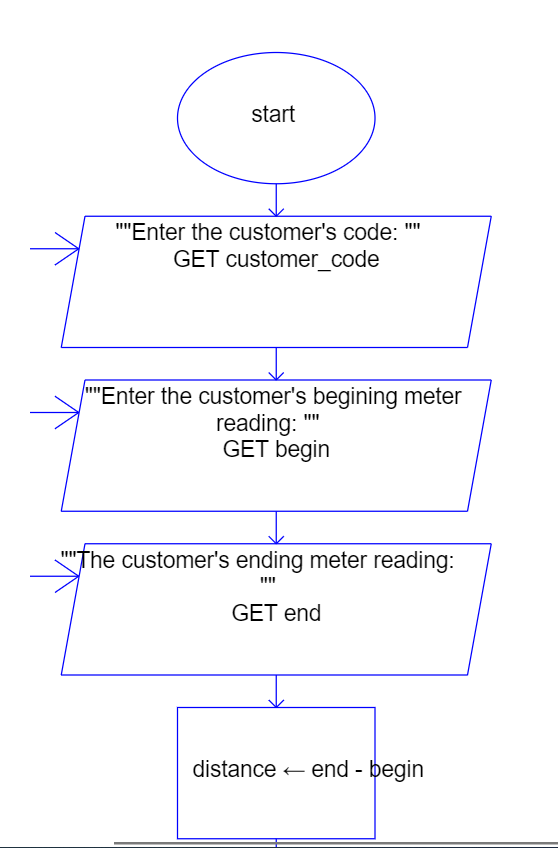
| **Input:** | **Process:** | | | |  | **Output:** |
| --- | --- | --- | --- | --- | --- | --- |
| **customer\_code** |  | **0<begin < 10e8 and 0<end<10e8** | **R or C or I or r or c or i** | **R or r** | **5 dollars plus 0.0005 per gallon used** | **bill = 5 + (0.0005 \* water\_g)** |
| **C or c** | **water\_g <= 4million** | **bill = 1000** |
| **water\_g > 4million**  **0.00025 for each extra gallon after 4milllion** | **bill = 1000 + (0.00025 \* (water\_g - (4 \* 10e5)))** |
| **I or i** | **water\_g less than 4 million** | **bill = 1000** |
| **water\_g between 4 and ten million** | **bill = 2000** |
| **0.00025 for each extra water\_g after 10million** | **bill = 2000 + (0.00025 \* (water\_g - 10e6))** |
| **Not (R or r or C or c or I or i)** | **water\_g =0** | | **'Invalid Entry'** |
| **begin** | **distance= end-begin**  **if end < begin:**  **distance= 10e8 - (begin-end)**  **water\_g = distance/10** |  |  |  |  | **"Customer code:", customer\_code**  **"Beginning meter reading:", begin**  **"Ending meter reading: ", end** |
| **end** |
|  |  | **begin and end are not between 0 and billion** |  |  |  | **'Invalid Entry'** |

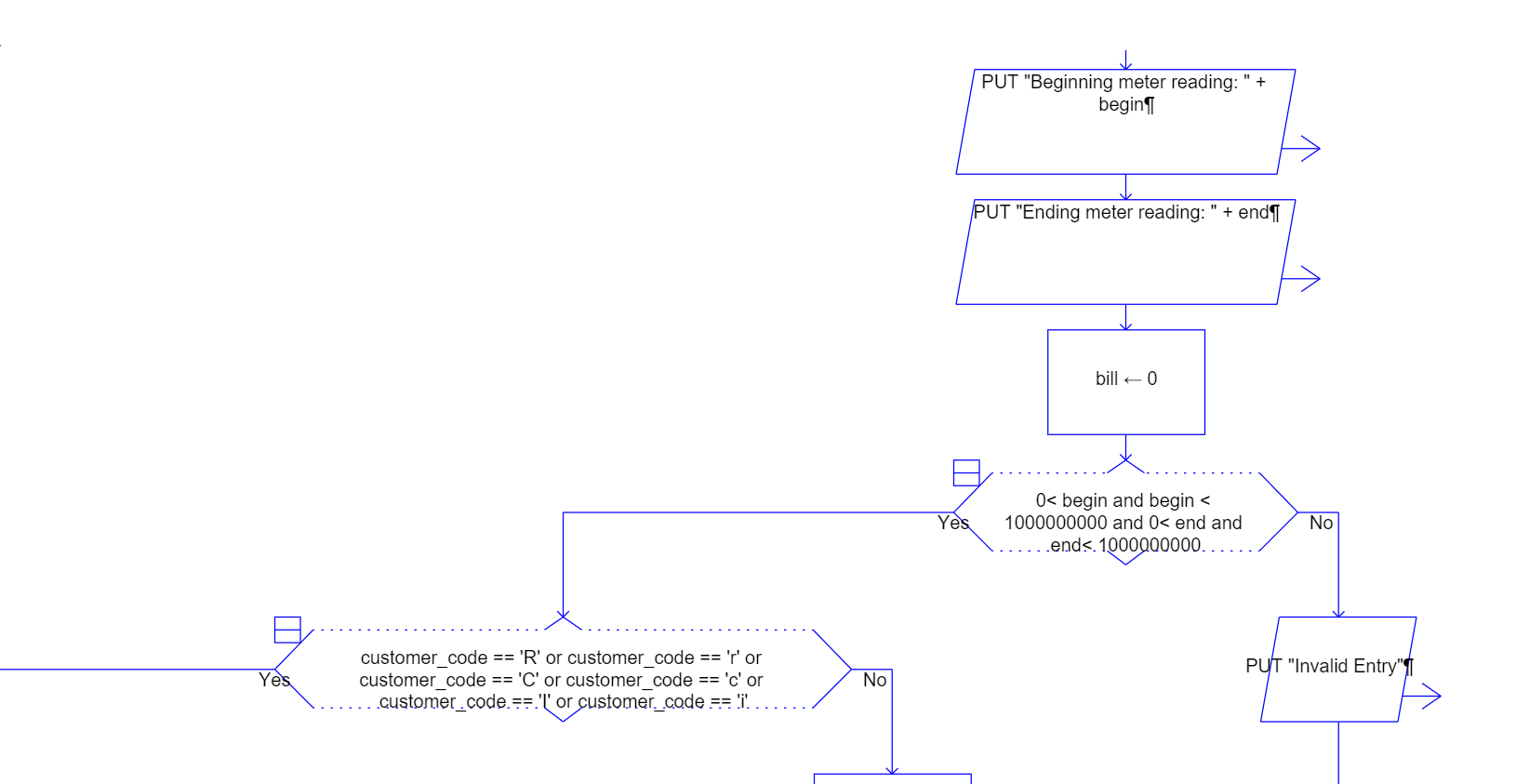
* 1. The **flowchart**

[Raptor file](https://drive.google.com/file/d/1qsdKUeOMAEP82Flglf_JWJnGcGzIrmMa/view?usp=sharing)

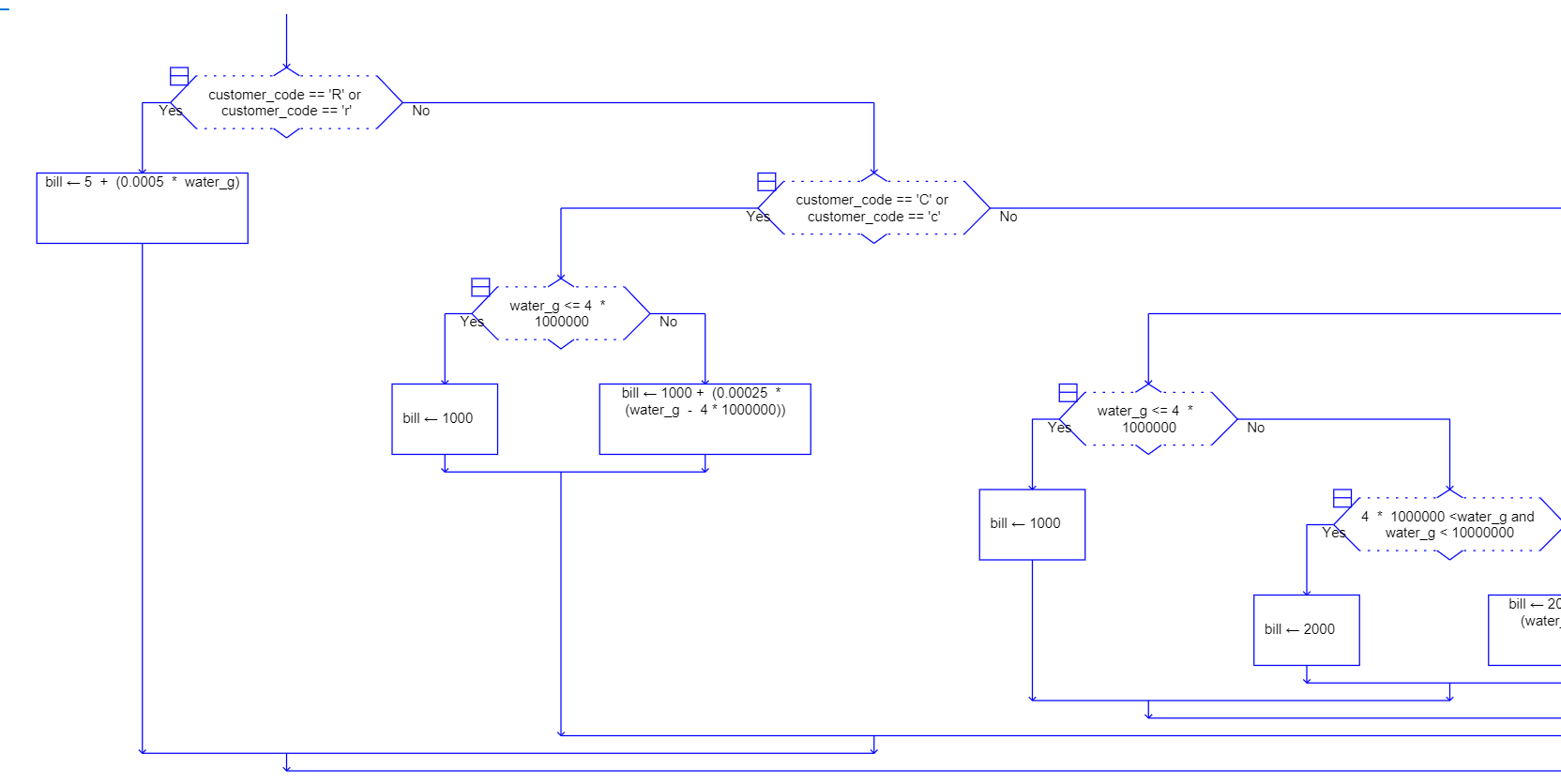
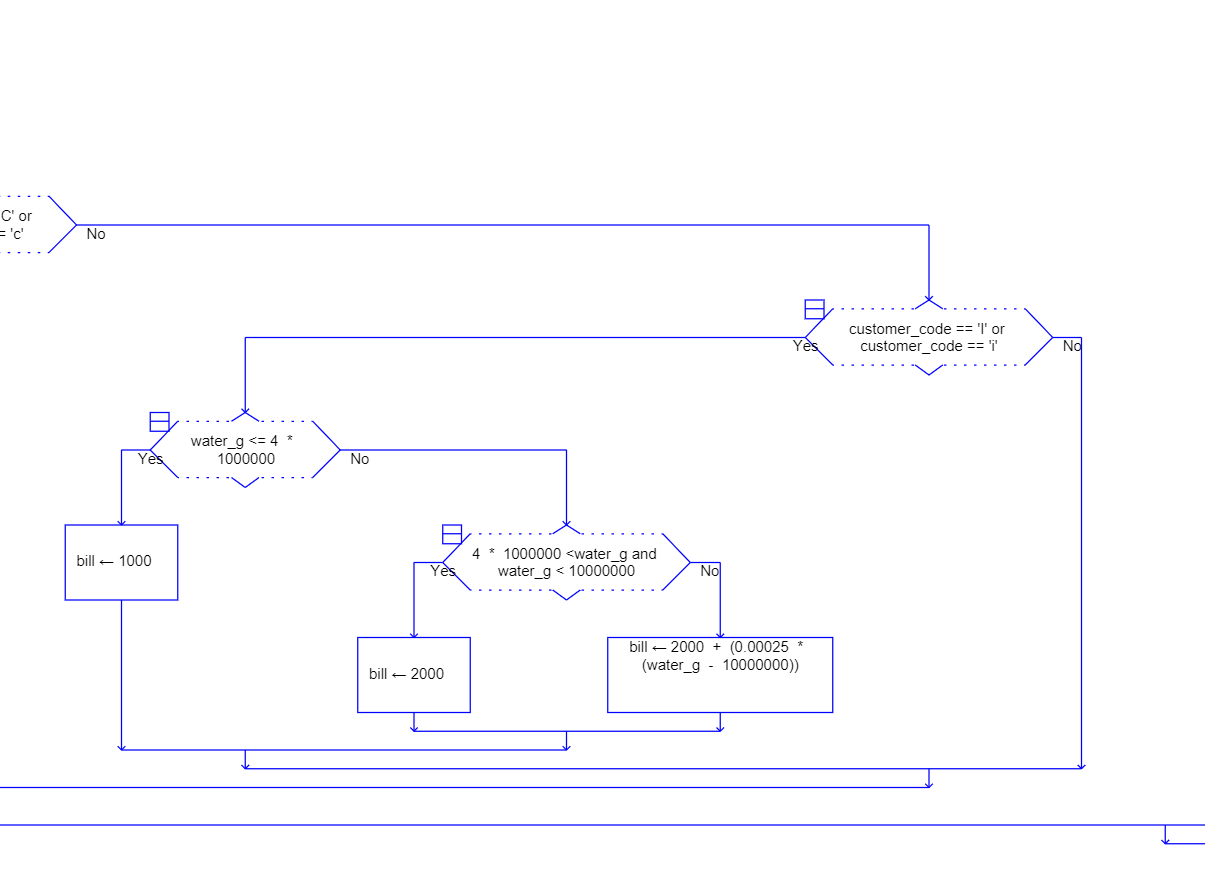
The flowchart at a glance



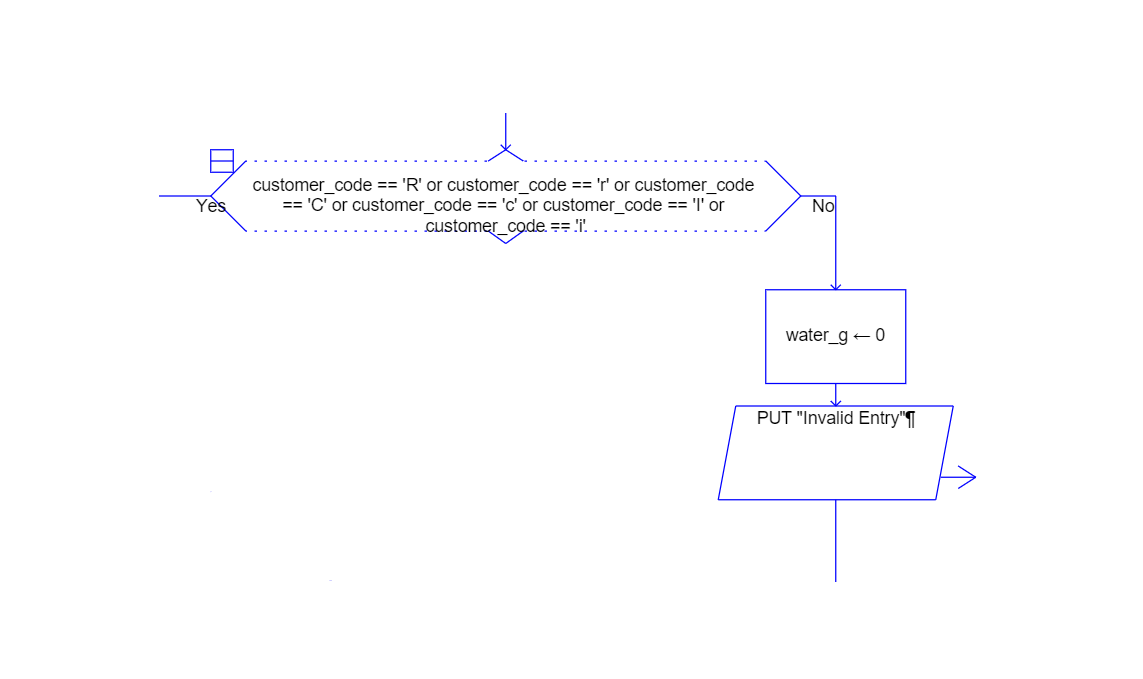


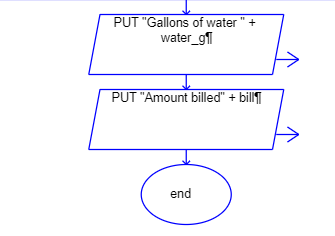


If (customer\_code == ‘R’ or customer\_code == ‘r’ or customer\_code == ‘C’ or customer\_code == ‘c’ or customer\_code == ‘I’ or customer\_code == ‘i’ ) == True/Yes => the flowchart below



If (customer\_code == ‘R’ or customer\_code == ‘r’ or customer\_code == ‘C’ or customer\_code == ‘c’ or customer\_code == ‘I’ or customer\_code == ‘i’ ) == False /No => the flowchart below

After all the conditions were checked::>



1. **Program Implementation:**
   1. The data type that we used was integers as the inputs that the user will input are integers, rather than floating point numbers. The operations that we implemented in our program were addition, subtraction, multiplication and division. Lastly, the built-in functions that we used were the import math, format(), input(), int(), and print(). The math module was used to find the absolute value and the format function was used to make values have 9 digits.
   2. What was challenging when we were implementing our program was to make sure the if states are in order as well as keeping track of the little details that have to be outputted. The order of the nested if statements matter as it is being checked in order. Moreover, little details such as dollar signs and having the outputs aligned was a concept in which we had to remind ourselves to do. Mathematically, making sure that the formulas are being written correctly was also challenging as the interpreter does not tell us if we have a logical error. The program can run smoothly, but it can still be wrong. Therefore, constantly checking it and revisiting the math was challenging in the program.
   3. One concept of the program that was straightforward and easy to implement was the print statements on what to output. Those are relatively straightforward as we are told what to output when asking for an integer. We just have to type it out correctly and that is it. Moreover, using the math module was simple to use as we have been practicing the use of it throughout the semester.
   4. We did not add more tests after we implemented the code. This was because we already thought of the types of inputs we were going to test during the implementation of the program. Although we did not test after the regular testing, we did test the program after a few days of not editing to double check if the program was still functioning as before.
   5. To make sure that the program still runs even when there is bad input input, we made sure that there were if statements for those inputs. The test would run, but will not do any calculations as those calculations are only restricted for inputs r,c, and i. Moreover, to prevent having a logical error, we used the absolute value function to make the calculations mathematically correct. Overall, the program will run and will produce the correct output for bad inputs.
   6. Bugs and/or Errors:
      1. Logical error: When being provided with an input where the ending was less than the beginning, it would create a negative value rather than a positive value. To fix this logical error, we implemented the use of the absolute value functions to always get a positive value.
      2. Syntax error: When formatting the values to have 9 digits, typing the format ("{:0>9}".format(begin)) was difficult as it was our first time using that format function. At first we implemented it with the f” “ string, but later changed it to a regular string “ “ and saw that it worked out better.
2. **Conclusion:**
   1. To sum, even though there were difficult logical errors in our program, debugging went well in our project. We were able to find the problem in our code and were able to run it correctly the second time without hesitation. Moreover, not forgetting about the little details about outputs went well as we took a deeper dive into reading the instructions to the project.
   2. Given another opportunity, we would write our code differently so as to not use a lot of if statements. Although that was required, using different types of code would be great as it will expand our knowledge about different and easier ways of programming. At the time we only had knowledge about if statements because that is what we learned. For future projects, narrowing down our code would be best as there would not be a lot to read and would not take a lot of lines. When it comes to time management, there is nothing that we would change as we put in our effort to complete the code as soon as possible and use the rest of the time to write this report.
   3. Improvements that might have made the project clearer would be to have a set of reminders at the end of the document that tells us to keep in mind the little details. For example, the program's correct output would have to be aligned, however, it was difficult to realize that until after completing and comparing our outputs side to side. Moreover, at the end of the document, it stated to submit only the screenshots of our programs code and output. Therefore, that led to some questions on whether or not we should have completed this report.
   4. Improvements that the instructions could have done to promote learning is to create a project that will be in interest for the students. Although this project about a water utility company was interesting, students would much rather prefer a project about their own interest such as social media, popular subjects, gaming, events, etc… This would just be the topic. The calculations and programming will remain the same level of difficulty.

**Appendix:**

[source code py file](https://drive.google.com/file/d/1WbQuWTbBfPvpJJP5jWQKMV1ObcKn3y9v/view?usp=sharing)

Project 2 Water Utility Company Source Code:

import math

''' The program will ask for three inputs,

including Customer's code, beginning meter reading and ending meter reading respectively'''

customer\_code = input("Enter the customer's code: ")

begin = int(input("Enter the customer's beginning meter reading: "))

end = int(input("The customer's ending meter reading: "))

# It will compute the gallons of water used by the customer

'''The meter has nine digits'''

distance = end - begin

if end < begin:

distance = 10e8 - math.fabs(end - begin)

water\_g = distance / 10

# compute the amount of money that the customer wil be billed, based on customer's code and water usage

'''

Code r => residential 5 dollars plus 0.0005 per gallon used

Code c=> commercial 1000.00 for 4 million gallons or less and 0.00025 for each extra gallon

Code i=> industrial 1000.00 less than 4 million gallons

2000.00 between 4 and ten

2000.00 + 0.00025 for each extra after 10

'''

print()

print("Customer code:", customer\_code)

print("Beginning meter reading:", "{:0>9}".format(begin))

print("Ending meter reading: ", "{:0>9}".format(end))

bill = 0

if (0 < begin < 10e8) and (0 < end < 10e8):

if customer\_code == 'R' or customer\_code == 'C' or customer\_code == 'I' or customer\_code == 'r' or customer\_code == 'c' or customer\_code == 'i':

if customer\_code == 'R' or customer\_code == 'r':

bill = 5 + (0.0005 \* water\_g)

elif customer\_code == 'C' or customer\_code == 'c':

if water\_g <= 4 \* 10e5:

bill = 1000

else:

bill = 1000 + (0.00025 \* (water\_g - (4 \* 10e5)))

elif customer\_code == 'I' or customer\_code == 'i':

if water\_g <= 4 \* 10e5:

bill = 1000

elif 4 \* 10e5 < water\_g < 10e6:

bill = 2000

else:

bill = 2000 + (0.00025 \* (water\_g - 10e6))

else:

water\_g = 0

print('Invalid Entry')

else:

water\_g = 0

print('Invalid Entry')

'''

Outputs:

Customer's code

begin

end

gallons of water

amount of money

'''

print("Gallons of water used:", f'{water\_g:.1f}')

print("Amount billed:", f'${bill:.2f}')

print()

Project 2 Water Utility Company Output:

