Problem Statement — B1

CPU Emulation

Computer programs are essentially a set of instructions that your CPU runs. Each instruction will manipulate the state of the machine; for example, one instruction might add two numbers together. The time to run an instruction can be measured in “cycles”. The more cycles a CPU can run, the more instructions it can execute.

Instruction ID	# of Cycles	Effect on Stored Integer
0	1	x = x + 1
1	2	x = x ∗ 2
2	3	x = x ∗ 5

Table 1. Instruction Set for the Wildcat 64

Listed above in Table 1 is the instruction set for the *Wildcat 64*’s CPU. Its CPU stores an integer in memory that is initialized to 0, and each instruction modifies the stored integer. Whenever an instruction is run, it has to use the specified number of cycles in Table 1 to run.

Your task is to emulate the *Wildcat 64* machine by replicating the behavior described above. Your program should take 3 instruction ids to execute consecutively in the order given. Your program should then output the stored integer after running the instruction(s). Finally, your program should also output how many cycles were used after running the instructions. When taking in the instruction ids, you may assume that the instruction id will always be 0, 1, or 2.

Example 1	Example 2
Input: Instruction 1: 0 **Input:** Instruction 2: 1 **Input:** Instruction 3: 2 **Output:** The stored integer is 10 **Output:** The number of cycles is 6	**Input:** Instruction 1: 2 **Input:** Instruction 2: 0 **Input:** Instruction 3: 2 **Output:** The stored integer is 5 **Output:** The number of cycles is 7

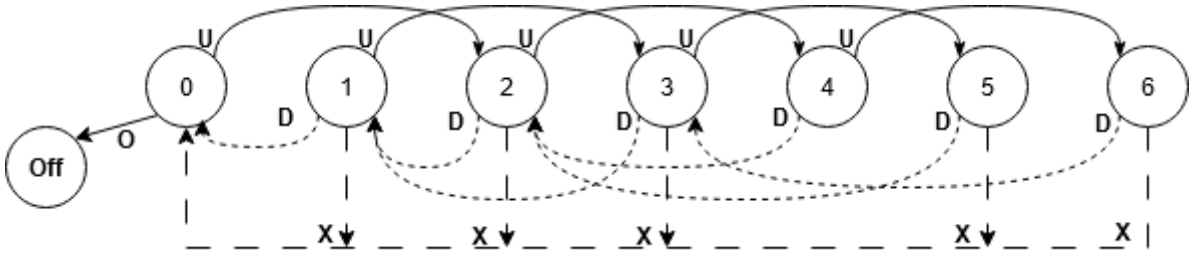
2 - Beginner - Gremlin Lifts

Problem Statement

Gremlin Lifts, a division of CBG Inc (Cheap But Glitchy Incorporated), has installed an elevator in a seven-story building. Due to their upbringing, Gremlin Lifts has numbered the floors 0 – 6 and refuse to re-number them, claiming that the request is “out of scope”. Gremlin Lifts, having been built by actual gremlins, don’t necessarily work the way one might hope.

The Gremlin Lift elevator has the same four 4 buttons in the elevator and on each floor: Up, Down, eXpress and Off. Using a proprietary technology, the elevator only responds to a button if it is already at a floor. Buttons pressed while the elevator is in motion are ignored.

Their transition from floor to floor is governed by the following diagram ... please note that Off is really the ground floor with the elevator turned off.



Fugre 1: Elevator Operational Diagram

Requirements

Buttons

- Pushing any button in the Off state has no effect—there is no on button. At installation the elevator was in the “0” state.
- The up button always causes the elevator to go up 2 floors, if it cannot go up 2 floors it does nothing.
- Pushing the down button causes the elevator to go down the half the number of floors it is on (rounded up). For example, if the elevator is on floor 5 and down is pressed, the elevator goes to floor 2 ($5/2$ rounded up is 3, so the elevator goes down 3 floors and ends stops on floor 2)
- The express button causes the elevator to go to the ground (0) floor, except a system glitch means the button does not work when on floor “4”.
- The off button only works if on the ground (0) floor. Any button pressed while the elevator is off will not work.

Input

- A string that contains any number of the following characters representing each button press: "D" for Down, "U" for Up, "X" for eXpress, and "O" for Off

Output

The program should output the final floor that the elevator is at once all button presses have been processed as well as the number of invalid button presses (button presses that caused nothing to happen) that were made.

Example

Input	Output
UUDUXUDO	Stopped on floor 3 Invalid button presses: 2
UUUDUDUX	Stopped on floor 4 Invalid button presses: 1
OUDU	Stopped on floor 0 Invalid button presses: 3

2023\gremlin_lifts\b2-gremlin-lifts-solution.py

```python
"""
Author: Josh Weese

Gremlin Lifts Beginner Model Solution
"""
import math
on = True
floor = 0
i = 0
invalid_count = 0
buttons = input("Enter the button presses: ")
while on and i < len(buttons):
    button = buttons[i].upper()
    i += 1
    if button == 'U' and floor < 5:
        floor += 2
    elif button == 'D' and floor != 0:
        floor -= math.ceil(floor/2)
    elif button == 'X' and floor != 4:
        floor = 0
    elif button == 'O' and floor == 0:
        floor = 0
        on = False
    else:
        invalid_count += 1
    print(f"At floor {floor} after {button}")
print(f'Stopped on floor {floor}')
print(f'Invalid button presses: {invalid_count + len(buttons) - i}')
```

3 Beginning — Divisibility
Problem Statement

A positive integer k is a *proper factor* of an integer n if:

- $k \neq n$ and
- k divides n; i.e., there is an integer i such that $ik = n$.

A positive integer n is:

- *deficient* if the sum of its proper factors is less than n;
- *perfect* if the sum of its proper factors is equal to n;
- *abundant* if the sum of its proper factors is greater than n.

Examples:

- 4 is deficient because $1 + 2 < 4$.
- 6 is perfect because $1 + 2 + 3 = 6$.
- 12 is abundant because $1 + 2 + 3 + 4 + 6 > 12$.

Write a program that takes as input a positive integer n and outputs whether n is deficient, perfect, or abundant. Note that there are at least two ways to determine whether k divides n using integer variables:

- See if $(n/k) * k = n$ (if k doesn't divide n, the integer division will round down to the next integer).
- Alternatively, see if the remainder when dividing n by k is 0.

Example 1:

```
Enter n: 4
4 is deficient.
```

Example 2:

```
Enter n: 6
6 is perfect.
```

Example 3:

```
Enter n: 12
12 is abundant.
```

```
1  // 3 Beginning - Divisibility
2
3  Console.Write("Enter n: ");
4  int n = Convert.ToInt32(Console.ReadLine());
5  int sum = 0;
6  for (int i = 1; i < n; i++)
7  {
8      if (n % i == 0)
9      {
10         sum += i;
11     }
12 }
13 if (sum < n)
14 {
15     Console.WriteLine(n + " is deficient.");
16 }
17 else if (sum == n)
18 {
19     Console.WriteLine(n + " is perfect.");
20 }
21 else
22 {
23     Console.WriteLine(n + " is abundant.");
24 }
25
```

4 - Beginner - Efficient Gardening

Problem Statement

Alice is a tech-savvy gardener who is looking to maximize the yield from her rectangular-shaped garden. She has information on various crops, including the spacing needed between plants and the expected yield per plant. Alice wants to use this information to decide which crop to plant in her garden to get the maximum yield.

Write a program that helps Alice determine which crop to plant in order to achieve the maximum yield. The program should also calculate how many plants can be planted in the garden for the selected crop and the total expected yield.

Input

- Two integers `L` and `W` ($L \geq 0, W \geq 0$): the length and width of the garden in meters.
- Three crops will always be entered. The following are entered for each crop: a string for the name of the crop, a float `S` (0.1 ≤ `S` ≤ 10) for the spacing between plants in meters, and a float `Y` (0.1 ≤ `Y` ≤ 10) for the expected yield per plant in kilograms.

Output

The program should output the best crop (the crop with the highest yield) along with the number of plants that can be planted on the field and the total expected yield in kilograms rounded to two decimal places. If there is a tie, only one needs outputted.

Other Constraints

- Each plant is planted directly in the center of a square space that is SxS where S is the specified spacing of that plant type. You may assume the garden is large enough to plant one of the crops given.
- The field can accommodate a whole number of plants. If the spacing causes a fractional number of plants to fit along the length or width, the number of plants should be floored to the nearest whole number.

Example

Input

```
10 10
Corn 1 2.5
Wheat 0.5 1.2
Rice 2 3
```

Output

```
Best crop is Wheat: 400 plants, 480.00 kg
```

2023\crop_calculator\b4-efficient-gardening-solution.py

```python
def calculate_yield(L, W, spacing, yield_per_plant):
    """Calculate the yield of a crop given the dimensions of the garden and the spacing between plants
    param L: length of the garden
    param W: width of the garden
    param spacing: spacing between plants
    param yield_per_plant: yield of the crop per plant
    return: the number of plants and the total yield
    """
    plants_along_length = int(L / spacing)
    plants_along_width = int(W / spacing)

    total_plants = plants_along_length * plants_along_width
    total_yield = total_plants * yield_per_plant

    return total_plants, total_yield

def read_input():
    """Read the input and return the data.

    Returns:
        tuple: The length and width of the garden, number of crops, and a list of crops.
        The list of crops is a dictionary containing the name, spacing, and yield per plant.
        Number of crops is fixed for this problem.
    """
    L, W = map(int, input('Enter field size: ').split())
    N = 3

    crops = []
    for i in range(N):
        crop_name, spacing, yield_per_plant = input(f"({i+1}) Enter crop_name, spacing, yield_per_plant: ").split()
        spacing = float(spacing)
        yield_per_plant = float(yield_per_plant)
        crops.append((crop_name, spacing, yield_per_plant))

    return L, W, N, crops


field_length, field_width, number_of_crops, crops = read_input()

# Keep track of the best yield and its corresponding data
best_yield = 0
best_crop_name = ''
best_plants = 0

results = []

for i in range(number_of_crops):
    crop_name = crops[i][0]
    spacing = crops[i][1]
    yield_per_plant = crops[i][2]
```

```
51
52      plants, yield_for_crop = calculate_yield(field_length, field_width, spacing,
    yield_per_plant)
53
54      # Check if this yield is better than the best so far
55      if yield_for_crop > best_yield:
56          best_yield = yield_for_crop
57          best_crop_name = crop_name
58          best_plants = plants
59
60      results.append((crop_name, plants, yield_for_crop))
61
62  # Print the results
63  print(f"Best crop is {best_crop_name}: {best_plants} plants, {best_yield:.2f} kg")
64  print(results)
```

B5 Minimizing Change in Pocket (CIP)

I don't like accumulating extra change in my pocket (CIP). Except for quarters, I want as few coins as possible. Design a calculator that keeps tracks of the CIP and when I have a bill to pay with cash, it suggests what change to give the clerk to minimize the change left in my pocket. In particular, I like to get rid of pennies and nickels.

To avoid some issues, we will not deal with dimes or dollar bills.

Assume you have sufficient change to pay off the change on the bill. Also, do not give the clerk more than 99 cents in change.

Example 1:

Input: Starting CIP: 8 pennies, 5 nickels, 4 quarters
The bill: $10.34

Output: Give 4 pennies, 1 nickel, 1 quarter

Example 2:

Input: Starting CIP: 8 pennies, 2 nickels, 1 quarters
The bill: $0.37

Output: 7 pennies, 1 nickels, 1 quarter

Example 3:

Input:
Starting CIP: 5 pennies, 8 nickels, 4 quarters
The bill: $30.87

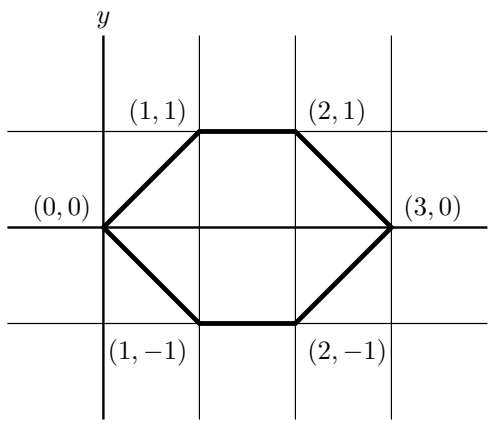
Output: 2 pennies, 7 nickel, 2 quarter

```
// B5 solution

#include <iostream>

int main()
{
    char quit = 'Q';
    while (quit == 'Q') {
        quit = 'N';
        double Bill;
        int Pchg, Bchg, Npennies, Nnickels, Nquarters;
        int Bpennies, Bnickels, Bquarters, Dollars;
        int UseP, UseN, UseQ;
        std::cout << "\nEnter the number of pennies in the pocket :";
        std::cin >> Npennies;
        std::cout << "\nEnter the number of nickles in the pocket :";
        std::cin >> Nnickels;
        std::cout << "\nEnter the number of quarters :";
        std::cin >> Nquarters;
        std::cout << "\nEnter the amount of the bill :";
        std::cin >> Bill;
        Dollars = Bill;
        Bchg = (100 * Bill) - (100 * Dollars);
        std::cout << "\ncents " << Bchg << "\n dollars " << Dollars;
        Pchg = 25 * Nquarters + 5 * Nnickels + Npennies;
        std::cout << "\n amount of change: " << Pchg << "\n";
        Bpennies = Bchg - (Bchg / 10) * 10;
        std::cout << "\npennies" << Bpennies; UseP = 0;
        if (Npennies > 0 && Bpennies > 0) {
            if (Bpennies <= Npennies)
            {
                UseP = Bpennies; Npennies = Npennies - UseP; Bchg = Bchg - UseP;
                std::cout << "\nResult1 UseP: " << UseP << " Bchg: " << Bchg;
            }
            else
            {
                if (Bpennies > 5 && (Bpennies - 5 <= Npennies))
                {
                    UseP = Bpennies - 5; Npennies = Npennies - UseP; Bchg = Bchg -
UseP;
                    std::cout << "\nResult2 UseP: " << UseP << " Bchg: " << Bchg;
                }
            }
        }
        //nickels
        UseN = 0;
        Bnickels = Bchg / 5;
        if (Bnickels > 0 && Nnickels > 0) {
            if (Nnickels >= Bnickels) {
                UseN = Bnickels;
```

```
            }
            else {
                if (Nnickels >= Bnickels - 5) { UseN = Bnickels - 5; }
                else {
                    if (Nnickels >= Bnickels - 10) { UseN = Bnickels - 10; }
                    else { if (Nnickels >= Bnickels - 15)   UseN = Bnickels - 15; }
                }
            }
            std::cout << "\nResult3 UseN: " << UseN << " Bchg: " << Bchg;
            Nnickels = Nnickels - UseN; Bchg = Bchg - 5 * UseN;
        }
        //quarters
        Bquarters = Bchg / 25;     UseQ = 0;
        if (Nquarters >= Bquarters) { UseQ = Bquarters; }
        Bchg = Bchg - 25 * UseQ; Nquarters = Nquarters - UseQ;
        Fquarters = Nquarters;
        std::cout << "\nResult4 UseQ: " << UseQ << " Bchg: " << Bchg;
        if (Bchg > 0) {
            Rchg = 100 - Bchg;
            std::cout << "\nResult7 Rchg: " << Rchg;
            Rquarters = Rchg / 25; Rchg = Rchg - 25 * Rquarters;
            std::cout << "\nResult8 Rquarters: " << Rquarters << " Rchg: " << Rchg;
            Rnickels = Rchg / 5; Rchg = Rchg - 5 * Rnickels;
            std::cout << "\nResult9 Rnickels: " << Rnickels << " Rchg: " << Rchg;
            Rpennies = Rchg;
        }
        std::cout << "\nquit?";    std::cin >> quit;
    }
}
```

6 Beginning — Hexagon
Problem Statement

In the hexagon shown to the right, the two horizontal edges have length 1, and each of the other four edges have length $\sqrt{2}$, which you should approximate as 1.414. We wish to travel clockwise along this hexagon, starting at point $(0, 0)$, and traveling a given distance. Write a program that takes as input a positive floating-point number d less than the perimeter of this hexagon, and produces as output the Cartesian coordinates of the point reached after traveling this distance from $(0, 0)$.



Example 1:

```
Enter d: .707
(0.5, 0.5)
```

Example 2:

```
Enter d: 2.414
(2.0000000000000004, 0.9999999999999997)
```

Example 3:

```
Enter d: 7
(0.46393210749646385, -0.46393210749646385)
```

Note: Your answers may not match the above exactly, but they should agree to three decimal places; for example, because the y-coordinate for Example 2 rounds to 1.000, the value produced should satisfy $0.9995 \leq y < 1.0005$.

```
// 6 Beginning - Hexagon

Console.Write("Enter d: ");
double d = Convert.ToDouble(Console.ReadLine());
double diag = 1.414;
double x, y;
if (d <= diag)
{
    x = d / diag;
    y = x;
}
else if (d <= 1 + diag)
{
    d -= diag;
    x = 1 + d;
    y = 1;
}
else if (d <= 1 + 2 * diag)
{
    d -= 1 + diag;
    x = 2 + d / diag;
    y = 1 - d / diag;
}
else if (d <= 1 + 3 * diag)
{
    d -= 1 + 2 * diag;
    x = 3 - d / diag;
    y = -d / diag;
}
else if (d <= 2 + 3 * diag)
{
    d -= 1 + 3 * diag;
    x = 2 - d;
    y = -1;
}
else
{
    d -= 2 + 3 * diag;
    x = 1 - d / diag;
    y = d / diag - 1;
}
Console.WriteLine("(" + x + ", " + y + ")");


```