Math topics 04

About my theory to explore this System indirectly😊(05/04/2022)

After writing about my point of view before and after the birth of this System, and most of you who know about me in Japan no longer trust this System. As a result, I almost didn't have to use my theory during my walk.😃This theory is for exploring this System indirectly. It's just around the right time, so I'd like to briefly explain this theory. 😃

Actually, when I was listening to the voices of this System's operators in my head five years ago, I was explaining this theory. At that time, I was afraid of this System, so I gave it to this System as a present for the time being.😃It seemed that my mentor was in this System. He was teaching to me a lot of important things at Graduate School of Management, Kyoto University. So I explained it to him. Now that I think about him, I feel ambiguous whether he is really my mentor himself.

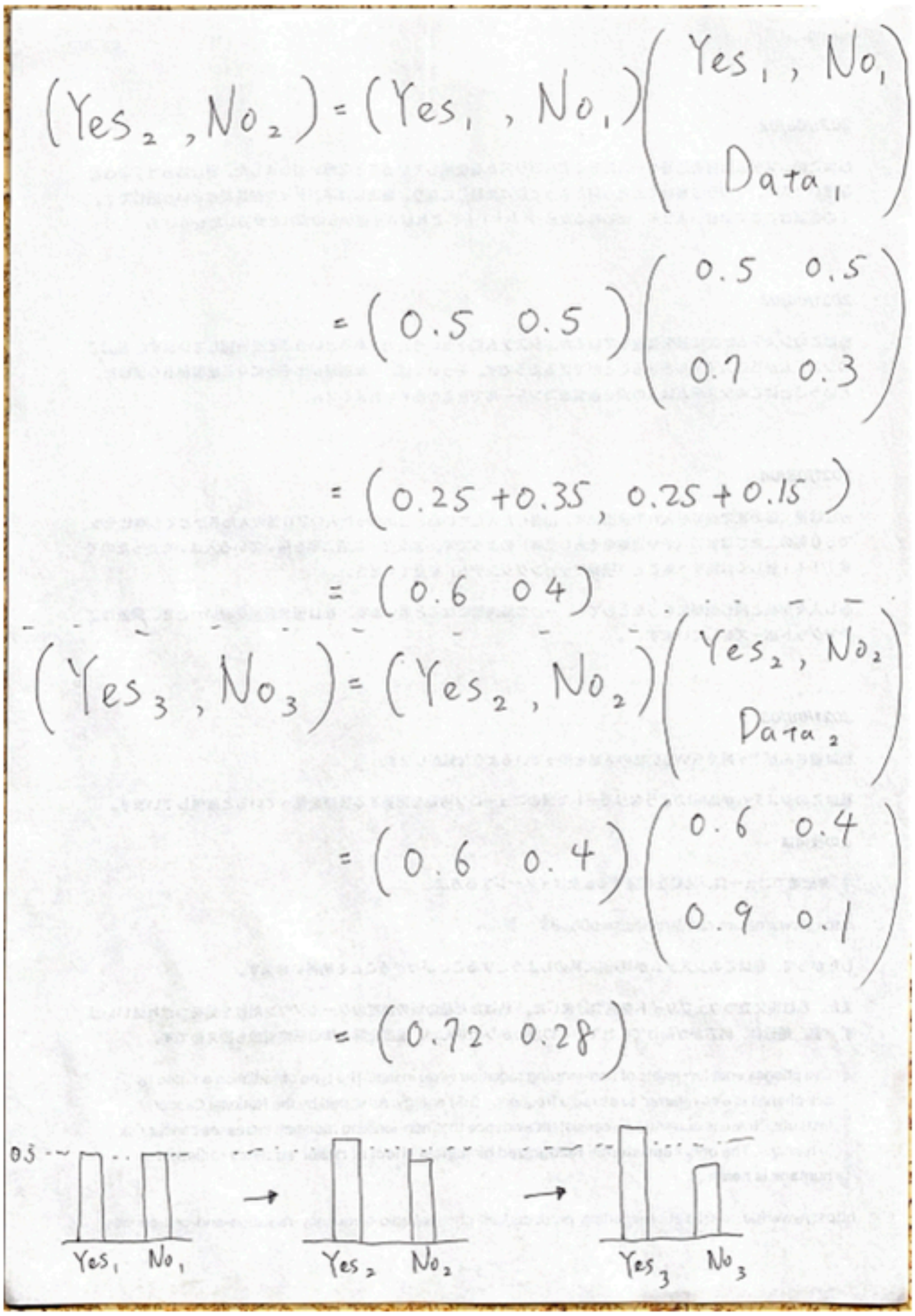
This theory uses something like the Markov Chain that I learned when I was assigned to his laboratory at Kyoto Graduate School (MBA). Simply speaking, the Markov chain can express a transition of a several-state using probability.

For example, I'm an actor, so in some places, I was looking at very conspicuous photos that I wouldn't tell people.😉 After that, let's check if this System is disclose it or not. There are two choices.

At the beginning, I don't know the movement of this System at all. So, whether this System is leaking that information or not, both will be 50%. It's the same as flipping a coin.

After that, I will get data such as a talk of a person who saw this System during a walk. Not everyone talks about that information. I will extract only stories related to my actions as much as possible. Next, I will decide the degree of data that is how much it has its relevance to my actions one by one, and change the probability of whether this System is leaking that information or not.

Let's say you get the first piece of data, and if there is a possibility that this System is leaking that information, it's a ratio 7:3. In other words, I use this information to update the ratio 5:5 of information above. Use a matrix to update it. I've written about "Transformation" using matrices in mathematical topics before, but here I will use it to update the information. In the picture below, I updated the vector like this:



$$(Yes_2, No_2) = (Yes_1, No_1)\begin{pmatrix} Yes_1, & No_1 \\ \multicolumn{2}{c}{Data_1} \end{pmatrix}$$

$$= \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}\begin{pmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{pmatrix}$$

$$= (0.25 + 0.35,\ 0.25 + 0.15)$$

$$= \begin{pmatrix} 0.6 & 0.4 \end{pmatrix}$$

$$(Yes_3, No_3) = (Yes_2, No_2)\begin{pmatrix} Yes_2, & No_2 \\ \multicolumn{2}{c}{Data_2} \end{pmatrix}$$

$$= \begin{pmatrix} 0.6 & 0.4 \end{pmatrix}\begin{pmatrix} 0.6 & 0.4 \\ 0.9 & 0.1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.72 & 0.28 \end{pmatrix}$$

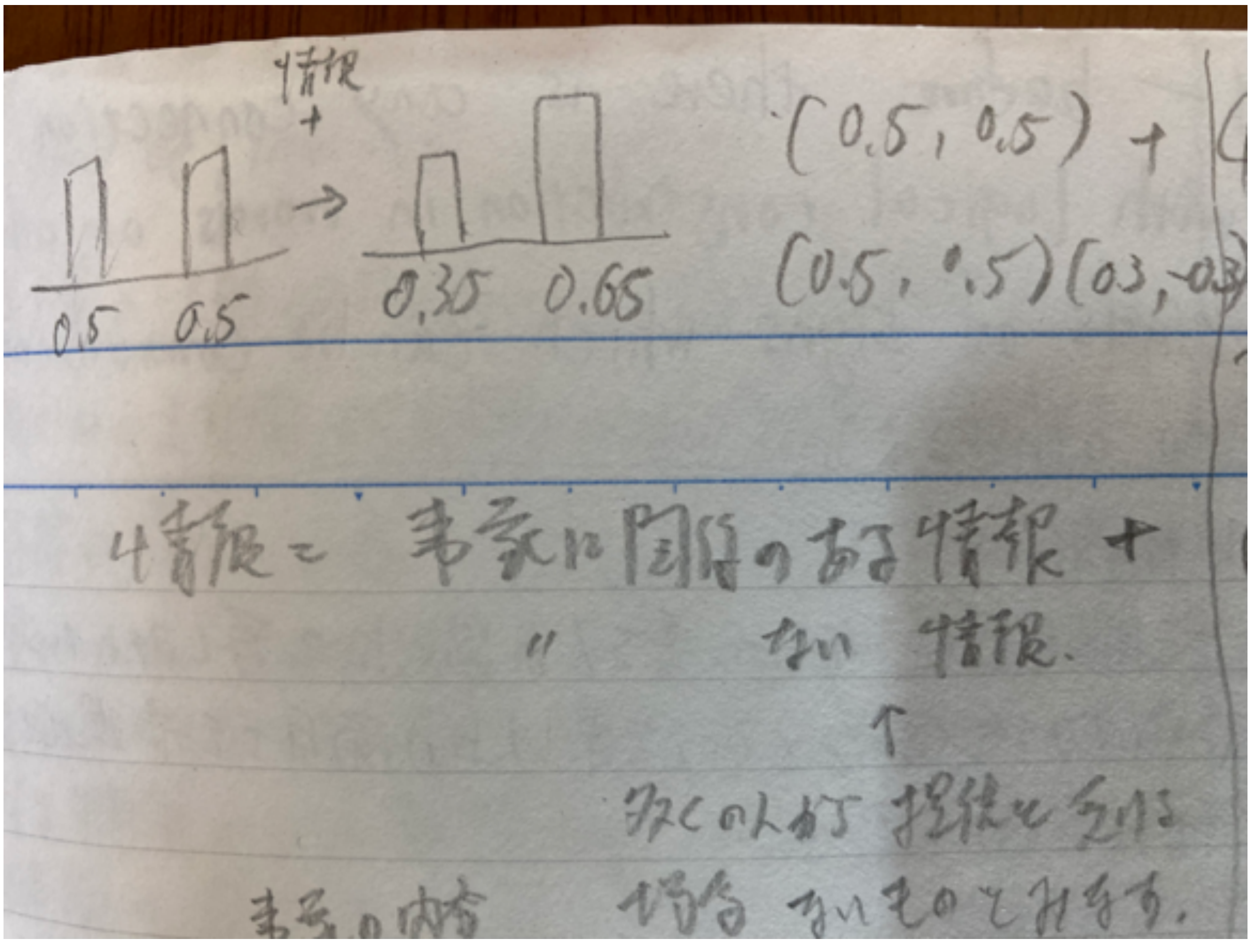
0.5
Yes_1 No_1
Yes_2 No_2
Yes_3 No_3

Suppose I get a small dataset such as (Data1) = (0.7, 0.3), (Data2) = (0.9, 0.1), I calculated it from (Yes1, No1) = (0.5, 0.5). So, I'll get (Yes2, No2) = (0.6, 0.4) and (Yes3, No3) = (0.72, 0.28) respectively.

Also, in the case of the Markov Chain, it is good to multiply a

same matrix, but in my theory, a matrix is different for each data, so commutative property of this multiplication is always not available, so if I change the order of the data, the result will be different.🤦‍♂️ There is a disadvantage of my theory.

Actually, when I explained it to this System five years ago, I didn't use formulas for its explanation.😉 I remember that I wrote it a little bit of my notebook at that time.



情報＝ 事象に関係のある情報 ＋
〃 ない 情報.
↑
多くの人が 推測と 分ける
事象の内容 場合 ないものとみなす.

I don't always do its calculations in my head according to my theory. If I explain it, it's just like that. It might be a good idea to finally take an average result to compensate for the above shortcomings.😉 Maybe there is a different theory of mathematics that the commutative property of this multiplication works.🧐

Keywords: Markov Chain, Probability, Matrix, Linear Algebra, Commutative Property

Self-reference has a mysterious power!? 🧐 (03/25/2022)

I was listening to the latest episode of the podcast called "Breaking Math". There were a lot of interesting stories about self-reference. Thank you very much.😊The following is one modified version of contents from this episode:

> The pie chart describes how many parts of the pie chart are red. In other words, this pie chart is a pie chart that explains its own chart.🙃

I was also able to learn Bootstrapping in the programming language.😃Also, there is a recursive function called its function within the function that often appears in programming languages.😃

I got to think that a basic mechanism of living organisms would be as simple as a recursive function. I'll briefly explain it on Swift Playground and Apple Script referring to an article called "4 Basic Phases to the Chicken Life Cycle".😃

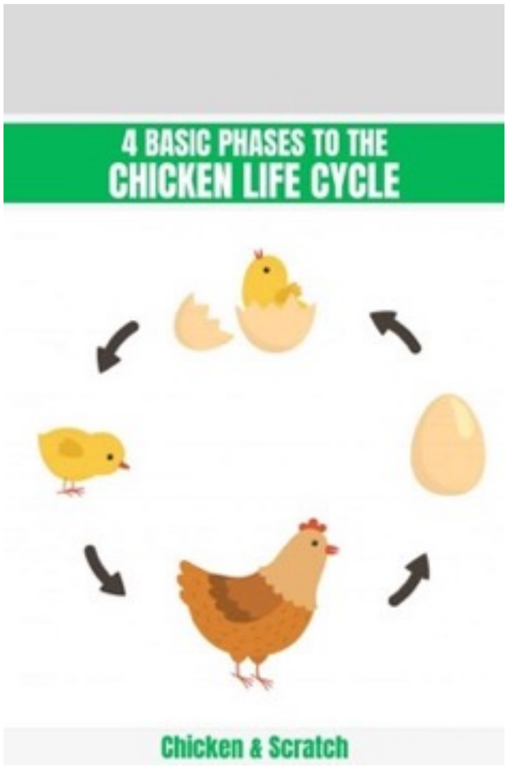
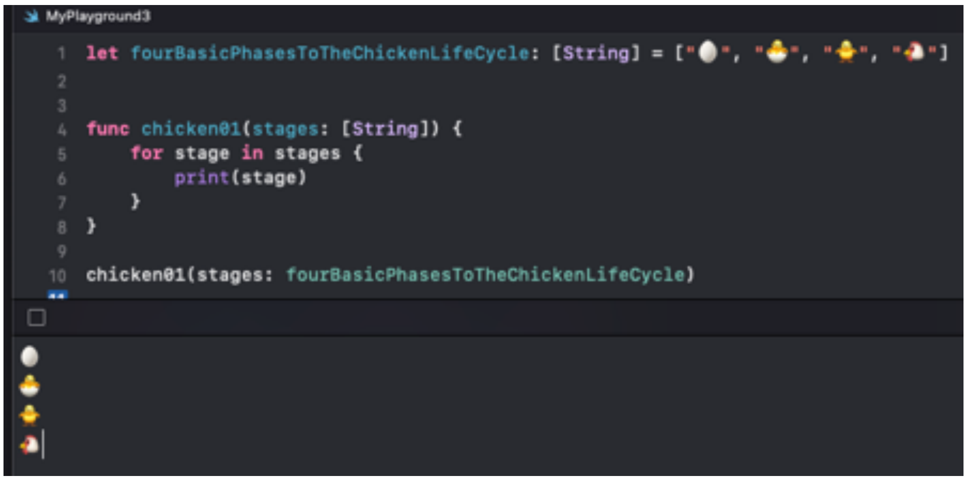


Image from the article

For example, if I express the life cycle of a chicken with four emojis that is (🥚,🐣,🐥,🐔), I made a chicken function that displays them one by one from the beginning. Its result was the image below. Also, with its Apple Script version, I made [a video]() to display emojis one second at a time.😃



```
let fourBasicPhasesToTheChickenLifeCycle: [String] = ["🥚", "🐣", "🐥", "🐔"]


func chicken01(stages: [String]) {
    for stage in stages {
        print(stage)
    }
}

chicken01(stages: fourBasicPhasesToTheChickenLifeCycle)
```

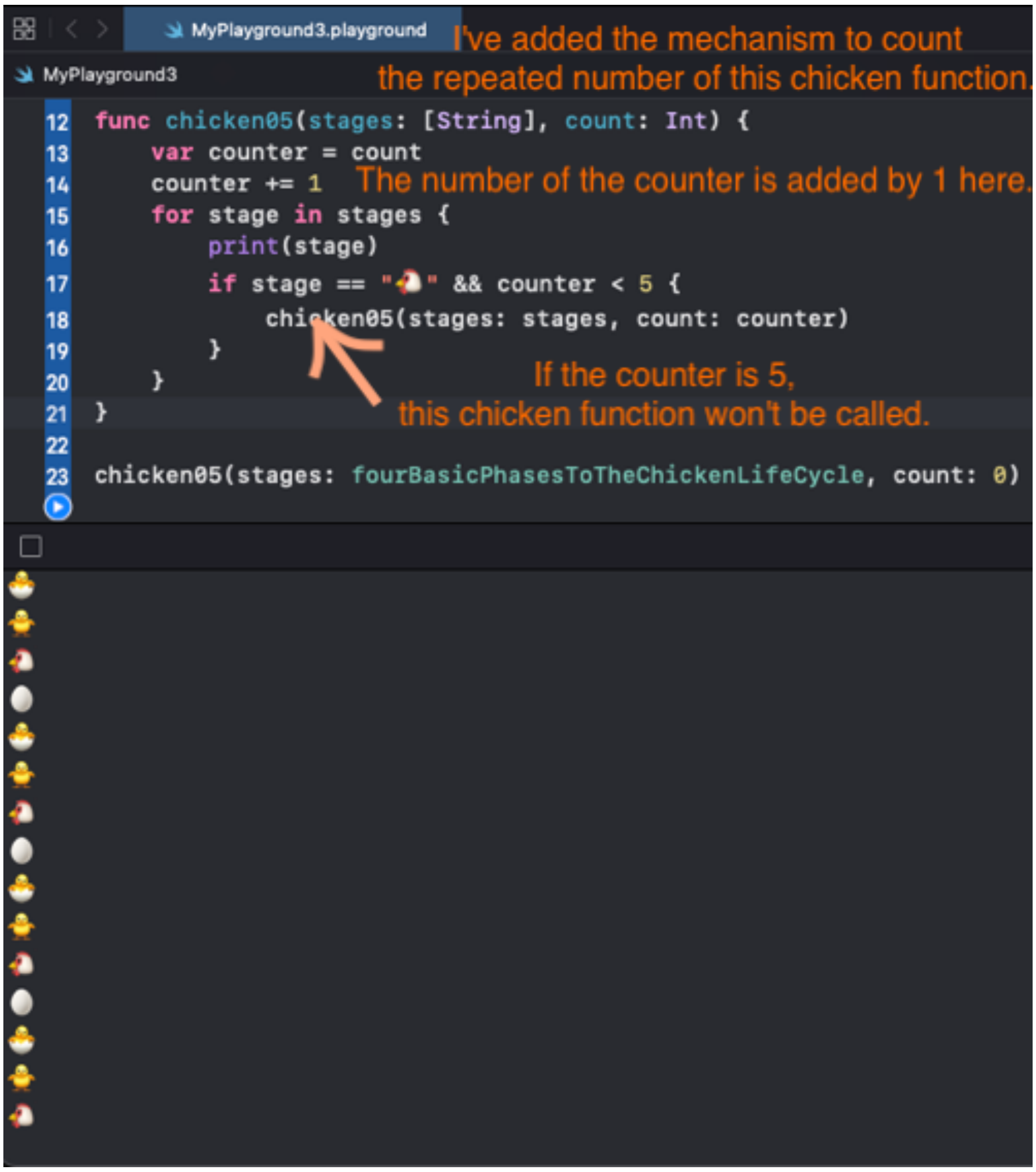
```
🥚
🐣
🐥
🐔
```

Image about the chicken function and its result

Let's put the chicken function itself in it. I'd like to explain it simply. Once I run it, it will be called to run while running it. You need to be careful here, because if you just put it in it simply, there is no mechanism for this function to stop, so it will continue to run. I somehow thought that living organisms are like this.

Then, I modified this function. It get to stop when it is iterated 5 times. Its result was the image below. **Note: you can see the part of its result.**

Also, I made another video with its Apple Script version.😃



I've added the mechanism to count the repeated number of this chicken function.

```swift
func chicken05(stages: [String], count: Int) {
    var counter = count
    counter += 1
    for stage in stages {
        print(stage)
        if stage == "🐔" && counter < 5 {
            chicken05(stages: stages, count: counter)
        }
    }
}

chicken05(stages: fourBasicPhasesToTheChickenLifeCycle, count: 0)
```

The number of the counter is added by 1 here.

If the counter is 5, this chicken function won't be called.

```
🐣
🐥
🐔
🥚
🐣
🐥
🐔
🥚
🐣
🐥
🐔
🥚
🐣
🐥
🐔
```

Image about the recursive function and the part of its result

Bonus:

Apple script can make a script read text, so I thought it could read emojis, so I tried it.👍In the Apple script, if I change the word "log" to the word "say", this script can also read emojis. 😊 You can easily do things like some kinds of accessibility feature.💪 But, the emojis don't represent exactly the chicken life cycle.😅

After I found Ron Garcia's tweet via Pikuma's Like on 04/20/2022, and it seemed that its content could be similar with my thoughts of a basic mechanism of living organisms. Thank you very much.😊

His tweet is an introduction to a part of a book called "Effective C", but it was easy to understand by showing examples of the precaution of for loop in C language.😃The following is a quote from an image of his tweet:

> Because of wraparound, an unsigned integer expression can never evaluate to less than 0. It's easy to lose track of this and implement comparisons that are always true or always false. **For example, the i in the following for loop can never take on a negative value, so this loop will never terminate:**

> for (unsigned int i = n; i >= 0; --1)

> This behavior has caused some notable real-world bugs. For example, all six power-generating systems on a Boeing 787

> are managed by a corresponding generator control unit. **Boeing's laboratory testing discovered that an internal software counter in the generator control unit wraps around after running continuously for 248 days, according to the Federal Aviation Administration. This defect causes all six generator control units on the engine-mounted generators to enter fail-safe mode at the same time.**
> To avoid unplanned behavior (such as having your airplane fall from the sky), it's important to check for wraparound by using the limits from ‹limits.h›. You should be careful when implementing these checks, because it is easy to make mistakes.

I looked up wraparound a little.😃An article on "Integer Overflow Attack and Prevention" was easy to understand. Thank you very much.😊The following are quotes from this article:

> **Integer overflow, also known as wraparound, occurs when an arithmetic operation outputs a numeric value that falls outside allocated memory space or overflows the range of the given value of the integer. Mostly in all programming languages, integers values are allocated limited bits of storage.**

> For example, we have a 16-bit integer value which may store an unsigned integer ranging from 0 to 65535, or signed integer ranging from -32768 to 32767. So, during an arithmetic operation, if the results require more than the allocated space (like 65535+1), the compiler may:

> *completely ignore the error caused, or
> *abort the program.

> Most compilers will ignore the overflow and store unexpected output or error. This will result in various attacks such buffer overflow which is the most common attack and leads to executing malicious programs or privilege escalation.

It was very helpful. Thank you very much.😊 I remembered the range of the specified values of integers in programming languages, But I didn't care about it recently. 🤦‍♂️

Keywords: Self-Reference, Pie Chart, Bootstrapping, Recursive Function, Mechanism, Living Organism, Wraparound, C language, For loop, Boeing 787, Overflow