Computer Programming by Chickens and Rabbits

Paul S. Wang, Sofpower.com

October 3, 2022

The modern computer brought us the information age and a digital universe. Because it's a *universal machine*, a machine that can be programmed to do anything *computable*, its impact on the world and civilizations everywhere is much more profound than any other machine.

Thus, a clear understanding of programming is important for *Computational Thinking* (CT). Here, we will give a simple explanation of programming with an easy-to-grasp example. An appreciation of program writing can show us many effective ways to solve problems that are useful in different areas, even our daily lives.

This post targets the most general audience. No technical background is needed to enjoy it. The post is part of our *Computational Thinking* blog (`computize.org`).

Programming Is Not Exotic

In fact, we are all familiar with programming. We follow TV programs or radio programs to watch/hear our favorite shows. When we go to concerts or music events we get programs. The recipes you follow when you cook are programs. Further, you may have written (or programmed) a few recipes yourself. Yet not all recipes have the same clarity.

Some authors foresee blind spots and difficulties, making their recipes easy to follow, instead of leaving more questions than answers. The good thing is that with practice and experience everyone can write better and better programs.

Want to try some programming? Don't be afraid. You can do it. It is not hard. We'll show you.

Programmers Think Differently

A *programmer* is one who creates a program by writing it down as a sequence of instructions.

A program has to be written in such a way that **anyone and everyone** can understand and follow the instructions precisely. In fact, a good program should be **foolproof**. Because ultimately, a program is for an idiotic machine (the computer) to execute.

As you can see, this requires programmers to adopt a different mindset than lay persons. A programmer needs to ensure that everything is perfectly clear and that nothing is left to chance. Hence, a program is one tedious step after another. For each and every fork in the road, a plan must be laid out ahead of time for each path.

Definition of A Computer Program

Input-processing-output

A computer program is a step-by-step procedure that can be carried out by a computer. We can specify the following criteria:

- **Finiteness**: The procedure consists of a finite number of steps and will always terminate in finite time.
- **Definiteness**: Each step is precisely, rigorously, and unambiguously specified.
- **Input**: The procedure may receive certain data from the outside as input. Possible values for the data may vary within limitations.

- **Output**: The procedure achieves specific actions and produces results as its output.
- **Effectiveness**: Each operation in the procedure is basic and clearly doable by a computer.

A Program for Chicken and Rabbits

In late primary school or early middle school, an often used example of applying mathematics is the *Chicken and Rabbit Problem*–Given the total number of heads (H) and total number of legs (L), figure out how many chickens and how many rabbits are in the yard.

For example, if there are 12 chickens and 7 rabbits then we have 19 heads ($H = 19$). Because each chicken has two legs and each rabbit 4, we have $24 + 28 = 52$ legs. That is $L = 52$. Thus, if we know how many chickens and how many rabbits, we can easily calculate the total number of heads and legs.

The reverse is a bit harder. The question is "Given the total number of heads and legs as known quantities, how do we find the correct number of chickens and rabbits?".

Here are some easy cases. If we know $H = 1$ and $L = 2$ then we have one chicken and no rabbit. If we know $H = 1$ and $L = 4$ then we have one rabbit and no chicken. And so on.

We'll illustrate program writing by creating a program to solve this specific problem. Let's name this program `CRprog`.

First thing, we'll look at the input and output of `CRprog`:

Input: The number of heads H and the number of legs L
Output: The number of chickens c and the number of rabbits r

This means the program `CRprog` will receive as input data H and L. Of course, these must be non-negative integers (whole numbers). Then through a sequence of steps, the program computes the correct values of c and r. Finally, it displays these values as output.

Computing c and r

Now its time to talk about how to solve this problem. That is how to compute c (number of chickens) and r (number of rabbits) from the given input H (number of heads) and L (number of legs).

Often there are more than one way to find the answers from the input. For our problem, one way is to use brute force–trying all possible values for r:

$$r = 0,\ r = 1,\ r = 2,\ r = 3,\ r = 4,\ ...,\ r = H$$

In other words, try "no rabbits" to "all rabbits" and we'll find the solution, sooner or later. Each trial value of r is only a **guess** that needs to be tested to see if it is right or wrong.

Brute force search, also known as *exhaustive search*, is a problem solving method that systematically generates and tests each and every possible solution. Obviously, the method has many applications.

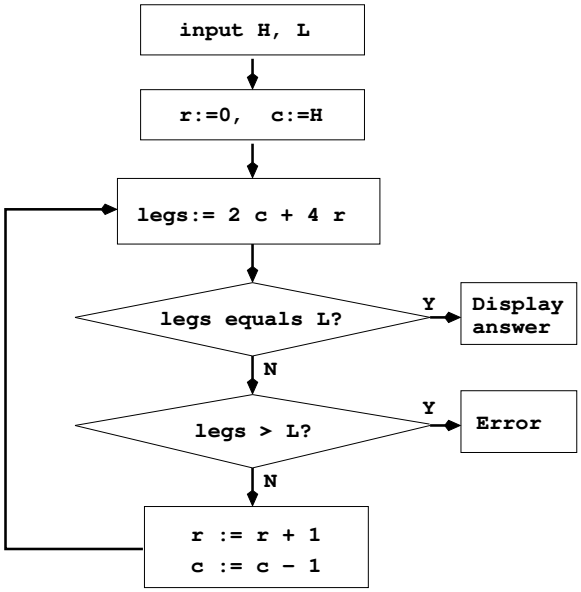
In the problem at hand, a guess r (number of rabbits) implies a corresponding guess $c = H - r$ (number of chickens), where H is the known head count.

To test the correctness of any pair r and c, we simply see if they lead to the given number of legs L. Specifically, we test if the following is true:

$$2 \times c + 4 \times r = L$$

Let's write down a flowchart for `CRprog` **brute force** that can make the whole process more obvious.

A Flowchart



In the flowchart we used the *assignment symbol* `:=` which gives the computed value on the right-hand side to the variable on the lefthand side.

In the flowchart you also see an *iteration*–a set of steps to be executed repeatedly. In a program, an iteration is also known as a *loop*. Using iteration is what makes trying all possibilities possible and simple to specify in a program.

Specifying A Sequence of Steps

Following this brute-force strategy and with the help of the flowchart, let's write down the sequence of steps for the program. The `comment` lines are for explanations and not part of the procedure instructions.

`CRprog` **brute force**:

1. Receive the input values H and L
   `comment`: H number of heads, L number of legs

2. Let $r := 0$; let $c := H$
   `comment`: r initial guess for rabbits, c for chickens

3. **repeat**: Let $legs := (2 \times c + 4 \times r)$
   `comment`: leg count based on current r and c

4. If $legs$ equals L then goto **answer**
   `comment`: if $legs$ equals the given L we found the answer

5. If $legs > L$ then goto **error**
   `comment`: if $legs$ is larger then L we have a problem

6. Let $r := r + 1$; let $c := c - 1$
   `comment`: for our next guess, add 1 to r, subtract 1 from c

7. goto **repeat**

8. **answer**: display "c Chickens and r Rabbits"; stop

9. **error**: display "no solution found"; stop

The best way to understand a procedure is to follow the *control flow* which flows from the current position/step to the next. Follow it from the very beginning all the way to the end. And you'll get a good idea how the procedure works.

At this time, if you are serious about learning how to program, or at least getting a good appreciation of it, you may want to try follow the steps using some specific input values, such as $H = 11$ and $L = 28$. See if you get the display "8 Chickens and 3 Rabbits".

Think of it as a game. Take out pencil and paper, carefully follow the procedure. If this is your very first time, congratulations! Because you have achieved a breakthrough not many have the chance.

Garbage In Garbage Out

Often, it is important for a program to validate its input data at the beginning. If the input is not valid then there is no point in going further.

Try `CRprog` **brute force** with the input $H = 0$ and $L = 0$. Does it work? This is known as an extreme case. How about $H = 9$ and $L = 15$? This input is invalid because L must be an even number. Of course any negative

or non-integral values for either H or L are invalid as well. Any L value less than $2H$ or more than $4H$ is also invalid. Can you think of other restrictions?

Please feel free to add these validity tests before step 1.

Program Efficiency

The brute-force method is not the best because it wastes computing power by trying too many guesses. Another initial guess could be r is about half of H. No matter what initial guess for r, we can compute $c = H - r$ and

$$legs = 2 \times c + 4 \times r$$
$$d = legs - L$$

If d is zero we found the solution. If d is positive then we have $d/2$ too many rabbits. If d is negative we have $-d/2$ too few rabbits. In either case $r + d/2$ is the correct number of rabbits, therefore the solution. This method takes just one guess.

But we can do even better and get the correct answer without any guessing. Using algebra on the next two equations

$$r + c = H$$
$$2 \times r + c = L/2$$

we have directly

$$r = L/2 - H$$

This means we can actually calculate the correct r and $c = H - r$ **in one step**.

Creating A Program In Stages

Given a problem to solve, we can follow a systematic approach to write a program for it.

1. Think about different ways to solve the problem. Figure out an overall solution strategy.

2. Use a flowchart to design the logical structure and the flow from one step to the next

3. Write down a procedure that checks the input, specifies and sequences the steps, and produces the answer
4. Test run the procedure on-paper with various input values, pay attention to extreme cases
5. Pick a computer language to code the procedure and do actual tests by running the code
6. Make corrections, improvements, and refinements.

An Improved Version

Let's now give the direct computation version of the chicken and rabbit program.

`CRprog` **direct**:

1. Receive the input values for heads and legs H and L
2. Check H and L for validity, if invalid goto **error**
3. Let $r := L/2 - H$; let $c := H - r$
4. Display "c Chickens and r Rabbits"; stop
5. **error**: display "Invalid input–no solution found"; stop

Coding

When we have the procedure figured out and precisely stated as we have demonstrated, then we are ready to put the program in a programming language of our choice.

Popular programming languages include C/C++/C#, Java, Perl, Python, Javascript, PhP, SQL, HTML, CSS, and many more. Some of them are *general-purpose* others are more specialized. Coding is an activity much like translation–take the step-by-step procedure given in English, or another natural language, and translate it into C++ or Python for example.

Once in the form of computer code, a program can be run on the target computer system for testing, debugging, revision, and updating. When ready, the program can be released for general use–until the next version takes its place.

Finally A Good Start

Through the chicken and rabbit problem we have introduced a complete example of program writing. You can see that both the brute-force version and the direct version pass the computer program criteria mentioned earlier.

The example also showed us how programming involves problem solving strategy, planning, anticipating problems, precise steps, power of iteration, logic and efficiency thinking.

The simple example here also demonstrated a systematic approach to programming: decide solution strategy, create flowchart, list step-by-step procedure, test, revise, code in a programming language, finally run the code, debug, and release.

We don't want to give the wrong impression that programming is an individual effort or it is so cut-and-dry. In fact, programming is an **art** that often also takes team work. Indeed, programming is a vast and deep area of computing and computer science, including software engineering, programming languages, compilers, algorithm design and analysis, application program interface (API) design, protocols, data structures, database, and much more. Proficiency in programming is highly valuable.

One simple example cannot make anyone a programmer. But you already had a peek into that world. Not everyone will like it. If you do, it can be a good start. Or at least it can add to your computational thinking.

By the way, there is a Javascript version of this program available online to try and download. If you are interested, please contact the author (email: pwang@cs.kent.edu).