



Grade 6 Math Circles

March 21/22/23, 2023

Control Flow

Controlling the Flow

In nature there are many things that flow. As humans, we try to control the flow so that we can harness that flowing energy and do stuff with it. Can you think of some examples of this?



Source: [Wikimedia Commons](#)



Source: [Wikimedia Commons](#)

An example of this is our electrical grid. These grids are full of power that we need to control the flow of. We can't just let the electricity run however it wants, because that would be disastrous and cause explosions. Our rivers are also an example of this. Rivers have flowing water, and sometimes we want to control the flow of the water so that it goes where we want it to go. As humans, we build dams or dig trenches to block or redirect the flow of water.

Control Flow in Computer Science

But how does this idea apply to computer science? In computer science, we use control flow as a way to make logical decision about what happens to an input variable. Before we dive in further, let's talk about some definitions.



Definitions

Pseudo-code: simplified code.

Variable: the name for an object that holds a value. Just like how x and y are variables in math and can take on values, the same can happen for variables in computer science.

If statement: a decision statement that is based on whether something is true or false. Sometimes this is called a **conditional statement**.

Switch case: a statement that re-directs the code to a different case based on multiple conditions.

Loop: a statement that controls when code is repeated. The two types of loops are *for loops* and *while loops*.

Break: a statement that exits the loop.

Example A

if statement:

Examples of if statements (in pseudo-code):

```
1 if(a=b) {
2     print("equal")
3 } else if(a>b) {
4     print("greater than")
5 } else {
6     print("less than")
7 }
```

```
1 if(a=b) {
2     print("equal")
3 }
4
5
6
7
```

The quotation marks show that the print statement gives us a word instead of just a number or another variable. Both sets of code have variables a and b . The code on the left compares their value before printing whether a is equal to b (on lines 1-2), greater than b (on lines 3-4), or less than b (on lines 5-6). The code on the right just compares a and b , prints whether they are equal or not, and does nothing otherwise.

switch case statement:

An example of a switch case statement:



```
1 switch(a) {
2     case 1:
3         print(1)
4     case 2:
5         print(2)
6     default:
7         print("Neither 1 or 2")
8 }
```

Case 1 is triggered when $a=1$, case 2 is triggered when $a=2$, and the default case is triggered otherwise. Note that a default case is always required! It's similar to an `if`, `else if`, and `else` statement, but not quite the same. We'll talk about this below.

for loop:

A `for` loop has this format:

```
for(variable with initial value; condition; increment/decrement).
```

We call this variable a counter variable, which can be increased or decreased. An example is the following code.

```
1 for(i=1; i<5; i++) {
2     print("Hi")
3 }
```

This code prints "Hi" four times. `i` starts at 1 and keeps looping until `i` is at least 5. `i++` increases `i` by one. We use `i` as our counter variable simply because it's common practice. There's actually no rule as to which variable you have to use.

while loop:

A `while` loop is just a loop with a condition. An example is the following code.



```
1 while(a=b) {  
2     print("Hi")  
3     a=keyboard input  
4 }
```

This code prints “Hi” as long as the condition, `a=b`, is true. If for example, `a=5` and `b=5`, then the condition would be initially true. If we do not give any keyboard input to change `a`, then the loop would continue forever. This is called an infinite loop.

break statement:

Suppose we started with `a` and `b` such that `a=b`. An example of a loop with a `break` statement is the following code.

```
1 while(true) {  
2     a=keyboard input  
3     if(not a=b) {  
4         break  
5     }  
6 }
```

Since the `while` loop’s condition is always `true`, then it would keep looping until the user inputs a value of `a` where `a` is different from `b`. When `a` is not equal to `b`, then the `if` statement will execute the `break` statement and exit the loop. This code is functionally the same as the example code on `while` loops.

Stop and Think

We just saw that an `if` statement and `switch case` statement are very similar. But how might we choose one or the other?

An `if` statement uses a conditional statement, which gives it the ability to compare different variables (such as `a>b`). A `switch case` statement just has an expression and different possible cases for the expression (such as `a=1` or `a=2`).

**Exercise 1**

Write a `for` loop that prints the numbers from 1 to 20. Then write the same thing but using a `while` loop.

Solution

For a `for` loop, we have the following code.

```
1 for(i=1; i<=20; i++) {  
2     print(i)  
3 }
```

For a `while` loop, we just make our own counter! We have the following code.

```
1 i=0  
2 while(i<20) {  
3     i++  
4     print(i)  
5 }
```

Notice that the initial value for `i` and the conditions are slightly different for both loops. But they both do the same thing since they will both run 20 times. Both loops increment `i` once per loop, but the `while` loop has to do it within the body of the loop instead of in the first line.

Example B

Suppose we wanted to write code for the system that determines whether a TV is turned on or off. The TV turns on whenever the user presses the on button and turns off whenever the user presses the off button. Also, when the TV is on, the TV will listen to which channel is being pressed and change the channel accordingly. The TV can access channels 25, 47, and 48. Other channels numbers will do nothing.

Solution



Since a TV is constantly “listening” to whether or not the remote’s ON button is being pressed, we want to loop this until the on button is pressed. In other words, we want loop listening to the button press while ON is NOT pressed. Let’s use `command` as the variable for the input.

```
1 while(not command=ON) {
2     command=button press
3 }
4 turn TV on
```

Line 4 says that we turn the TV on after the we exit the `while` loop.

Now, since our TV is ON, we want to “listen” to whether or not the remote’s OFF button is being pressed or whether the user wants to access a channel. Since we have multiple cases, we would want to use a `switch case` statement. So we have the following code.





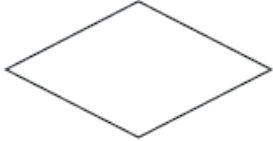
```
5 while(true) {
6     command=button press
7     switch(command) {
8         case OFF:
9             break
10        case 25:
11            change to channel 25
12        case 47:
13            change to channel 47
14        case 48:
15            change to channel 48
16        default:
17            do nothing
18    }
19 }
20 turn TV off
```

Lines 8 and 9 say that when the OFF button is pressed, we break out of the infinite loop and go to line 20, which turns the TV off.



Control Flow Diagrams

A very useful tool is a control flow diagram or flow chart that helps us to map out how a piece pseudo-code works. We can use it to map out the possible outcomes and logic within the code. Let's define some symbols for flow charts.

Symbol	Name	Purpose
	Start/End	Indicates the start or end of the diagram.
	Arrow	Shows the relationship between the different symbols.
	Input/Output	Indicates the input or output of a process.
	Process	Indicates the process in a diagram.
	Decision	Indicates where a decision needs to be made. In a diagram, this would be used for our <code>if</code> or <code>break</code> statements.

There are many more symbols that we will not talk about in this lesson. If you're interested, you can check them out [here](#).



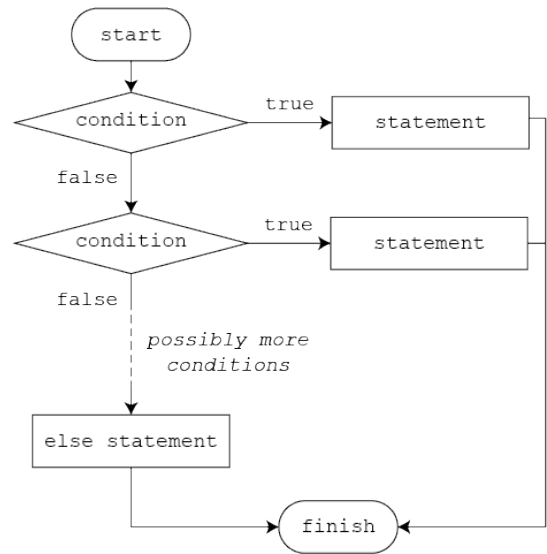
Let's look at how we can model if statements, switch case statements, for loops, and while loops with control flow diagrams.

if statements

```

1  if(condition) {
2      statement
3  } else if(condition) {
4      statement
5  } ... {
6      statement
7  } else {
8      else statement
9  }

```

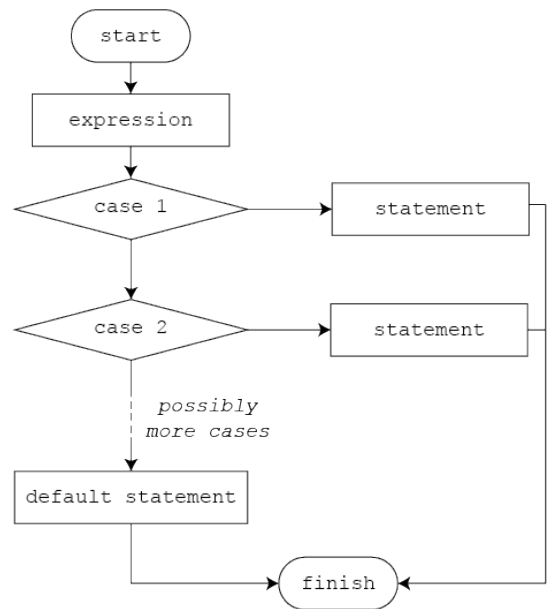


switch case statements

```

1  switch(expression) {
2      case 1:
3          statement
4      case 2:
5          statement
6      ...
7      default:
8          default statement
9  }

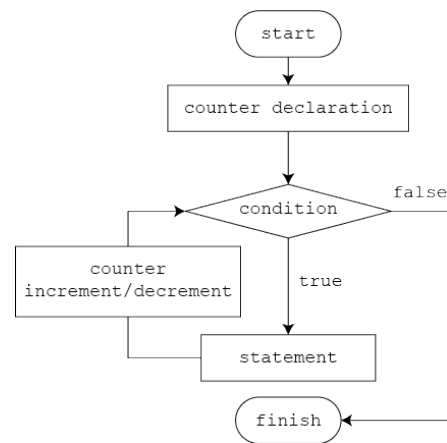
```





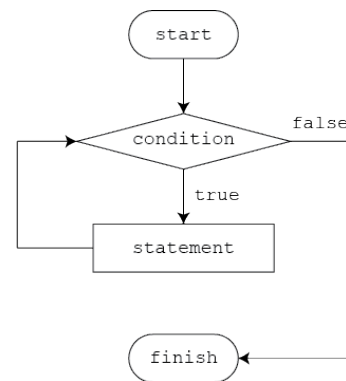
for loops

```
1 for(counter; condition; counter  
   increase or decrease) {  
2   statement  
3 }
```



while loops

```
1 while(condition) {  
2   statement  
3 }
```



Example C

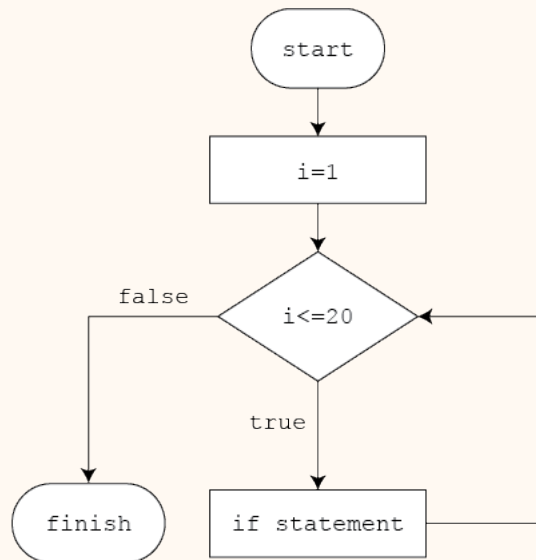
Suppose we had the following code and wanted to draw a control flow diagram for it. Note that `is_even(i)` returns `true` if `i` is even and `false` if `i` is odd, and `is_divisible_by_3(i)` returns `true` if `i` is divisible by 3 (has a remainder of 0 when divided by 3) and `false` if `i` is not divisible by 3 (has a remainder of 1 or 2 when divided by 3).



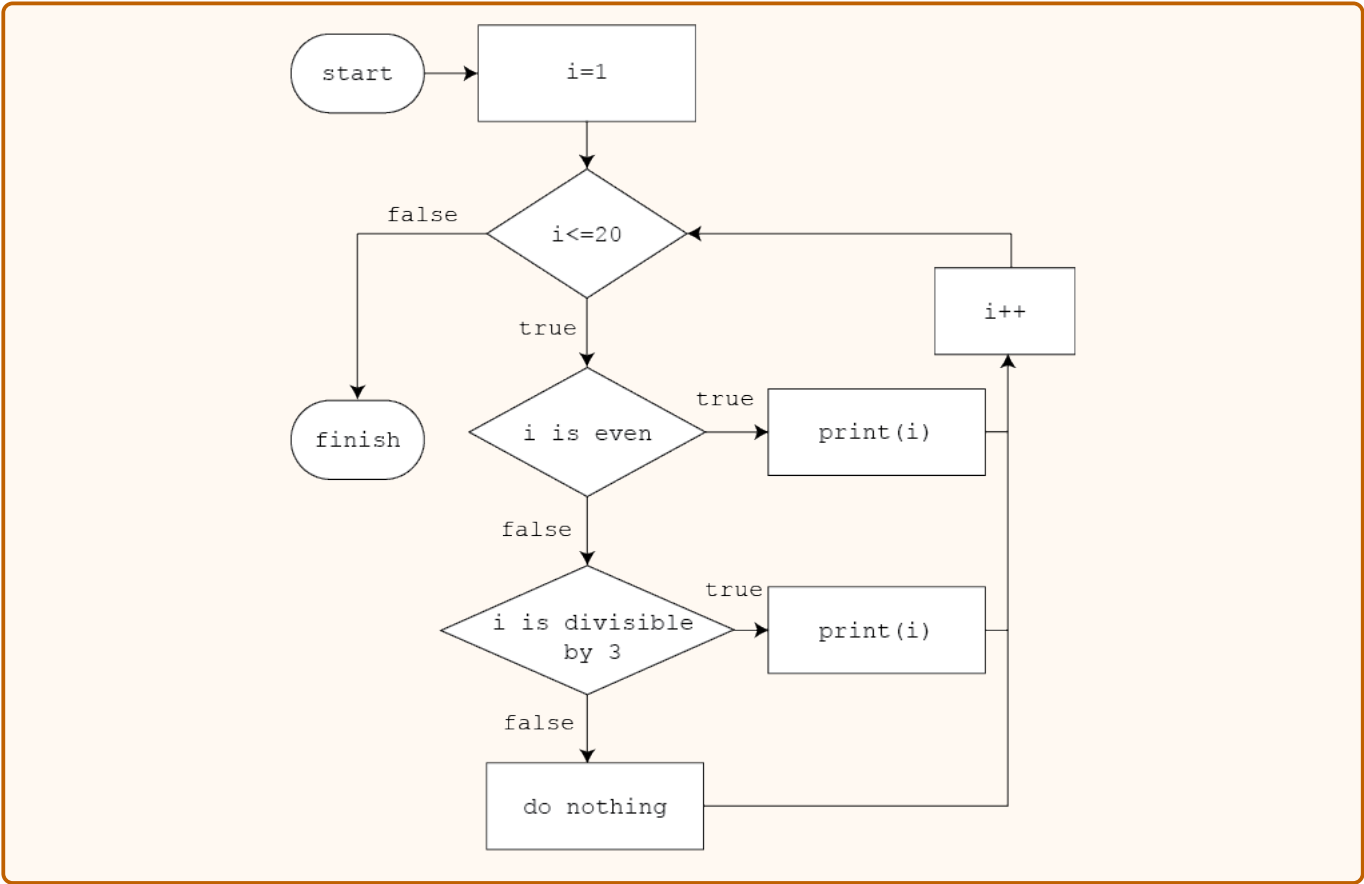
```
1 for(i=1; i<=20; i++) {  
2     if (is_even(i)) {  
3         print(i)  
4     } else if is_divisible_by_3(i) {  
5         print(i)  
6     } else {  
7         do nothing  
8     }  
9 }
```

Solution

This `for` starts with the variable `i` set to the value of 1 and has a conditional statement of whether `i` is less than or equal to 20. If this is `true`, then we enter the loop and have our `if` statement. So this would give us the initial drawing:



Every time we enter the loop, we `print(i)`, which is an output, and then increase `i` by 1 before looping back to the conditional statement. So now we get the final drawing:





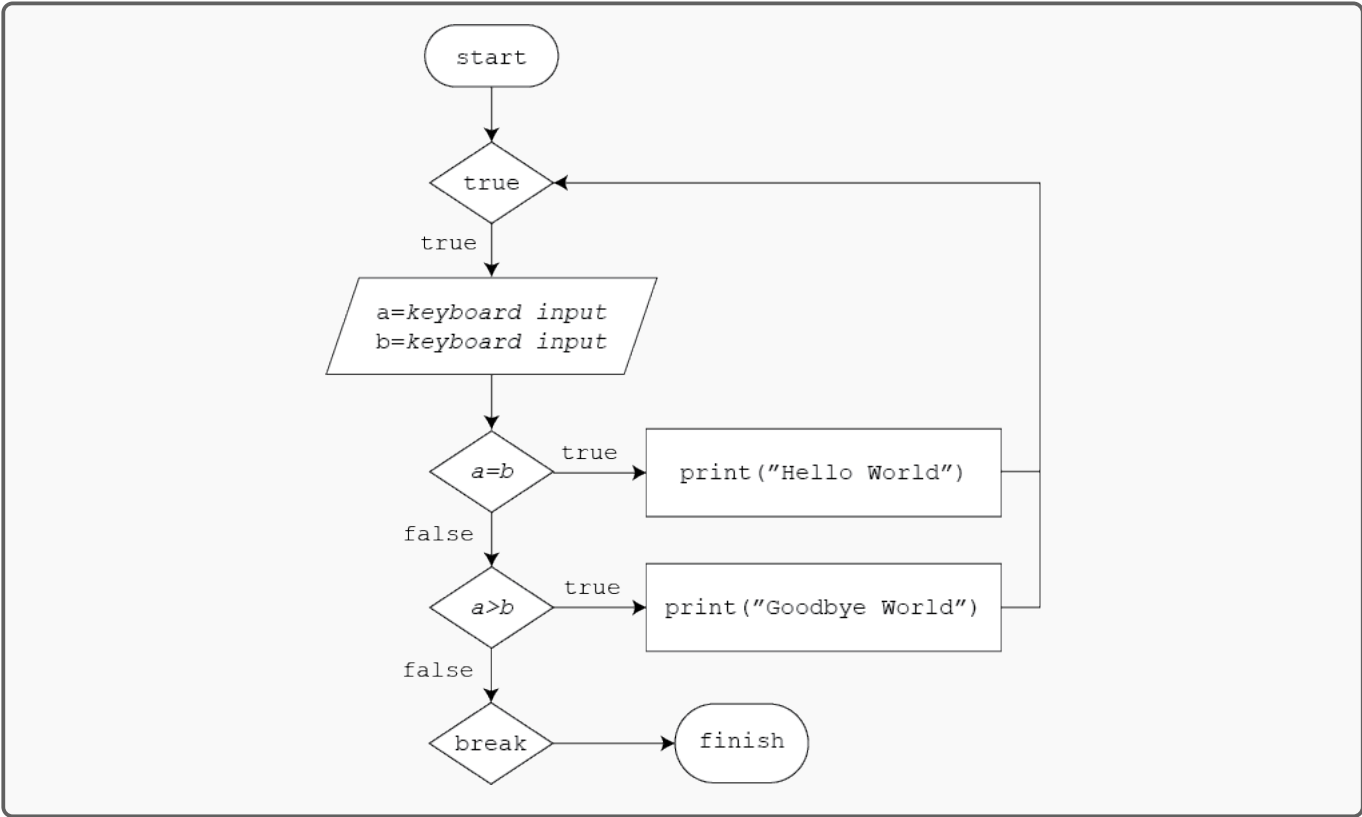
Exercise 2

Given the following code, draw a control flow diagram.

```
1 while(true) {
2     a=keyboard input
3     b=keyboard input
4     if(a=b) {
5         print("Hello world")
6     } else if(a>b) {
7         print("Goodbye world")
8     } else {
9         break
10    }
11 }
```

Solution

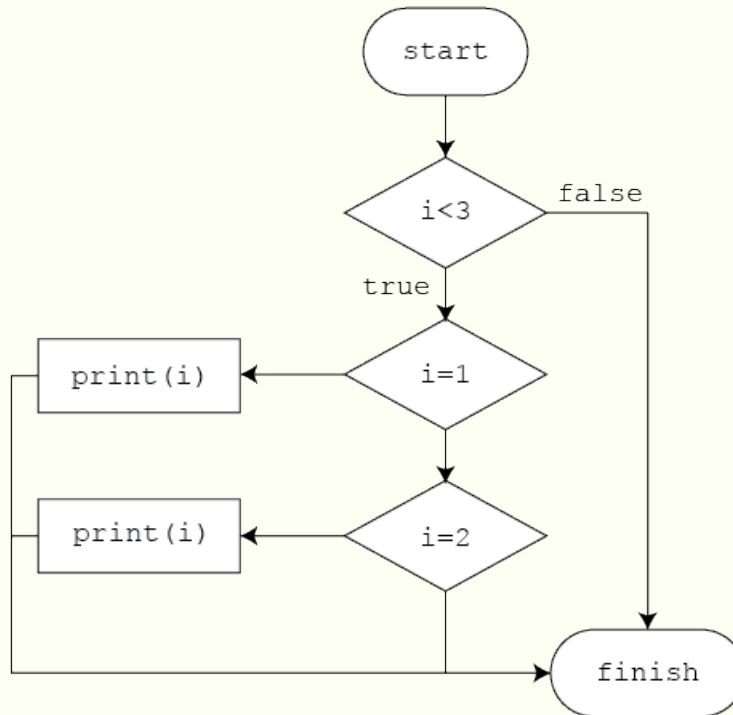
Even though the condition of the `while` loop is `true`, it is still a condition that will be included in the diagram. But since the condition is always `true`, we don't necessarily need to include a `false` option. Since `a` and `b` both take keyboard inputs, they will need an input symbol. We can combine both of these into one symbol. Our `if` statement has two conditions (the initial `if` statement and then the `else if` statement), so we need to reflect that in our diagram as well.





Exercise 3

Given the following control flow diagram, write code that matches it.



Solution

The first condition is whether `i` is less than 3. Since there is a `true` and `false` label, it is an `if` statement and not a `switch case` statement. It also isn't a `for` or `while` statement because we can see that the arrows don't flow in a loop (the arrows only head towards `finish`). There does not need to be an `else if` or `else` statement, because there are no other conditions to consider. Then the following conditions (`i=1` and `i=2`) must be `switch case` statements since there are no `true` or `false` labels. So we get the following code.



```
1  if(i<3) {  
2      switch(i) {  
3          case 1:  
4              print(i)  
5          case 2:  
6              print(i)  
7          default:  
8                
9      }  
10 }
```



Modeling Algorithms

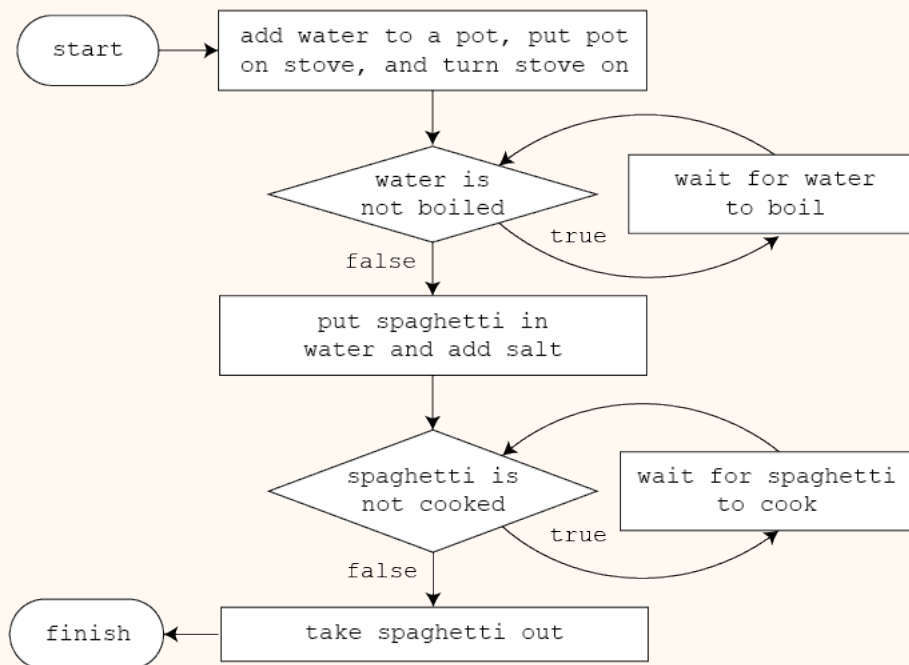
What is an algorithm? An algorithm is essentially a list of steps that help us solve something. You can think of it as a recipe, and we can model them with a flow diagram.

Example D

Let's consider the steps to cooking spaghetti. The (rough) instructions are as follows:

1. Add water to a pot, put it on a stove, and turn the stove to high
2. When the water is boiled, proceed to the next step.
3. Put the spaghetti in the boiled water and add salt.
4. After a few minutes, check on the spaghetti.
5. When the spaghetti is cooked, take it out.

So we can model these instructions with the following flow diagram.



Since we have a flow diagram, we can use our skills that we practiced before to also turn this into pseudo-code. We can model the two loops using a while loop, since there is no need to use a counter variable. So we have the following code.



```
1  add water to pot
2  put pot on stove
3  turn stove on
4  while(water not boiled) {
5      wait for water to boil
6  }
7  put spaghetti in water
8  add salt to water
9  while(spaghetti not cooked) {
10     wait for spaghetti to cook
11 }
12 take spaghetti out
```

Following the flow diagram or the pseudo-code will allow you to make spaghetti (successfully) every time. (Milage may vary.)

Exercise 4

Consider the following algorithm.

1. The user inputs any given natural number (positive whole number), which we will call n .
2. If n is even, then divide it by 2. If n is odd, then multiply it by 3 and then add 1.
3. Repeat step 2 until n is one.
4. Output the number of steps the number, n takes to reach 1.

Write code for this algorithm and draw a control flow diagram to match your code (or vice versa). Use `is_even(n)` as the condition for whether n is even.

Solution

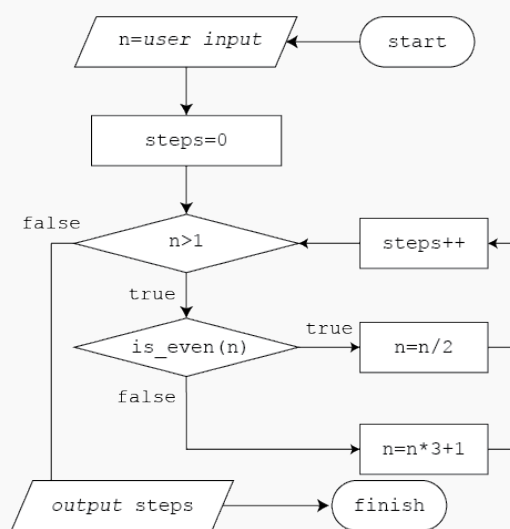
For this solution, we will use a `while` loop. We want to use our counter variable, which we will call `steps` at 0, because if n is already 0, the loop will not be entered, and it should have taken 0 steps to go from 1 to 1.



Note that in order to output *steps*, we need to use a *while* loop since *steps* would be a local variable to the *for* loop. Don't worry about this, since it's not something we are touching on.

Within the loop, we have two conditions; whether *n* is odd or even. But notice that since a number must be either odd or even, we can have an even condition and an *else* condition as the odd condition. So we have the following code and flow diagram.

```
1 n=user input
2 steps=0
3
4 while(n>1) {
5     if(is_even(n)) {
6         n = n/2
7     } else {
8         n = 3*n+1
9     }
10    steps++
11 }
12
13 output steps
```



Fun Fact!

This problem is commonly known as the $3n + 1$ (three n plus one) problem, or also called the Collatz conjecture. The conjecture is based on whether or not for *every natural number* as input will the sequence eventually reach one. In computer science terms, we can think of it as whether or not our program will ever end (or terminate). This problem remains one of the most famous unsolved math problems to this day. Computers have shown that many large numbers do eventually end up at 1, but maybe there's an even bigger number out there that doesn't terminate.

If you want to learn more about this, click [here](#).