Tips

1. In a while loop, where the **loop update** is done before the **loop body**, **the second part of loop condition** is **inclusive.**
2. An **assertion** is**: a logical expression that must always be true for the program to be correct.**
3. General Form of Assertion for "if-else" structure
    a. The if-else structure is **complete**. (a, b, X, Y, Z are all logical expressions)

```
if (a) {
  // { X };
} else if (b) {
  // { Y };
} else {
  // { Z };
}
// { ?? }
```

    The assertion we can make at // { ?? } is

```
// { X || Y || Z }
```

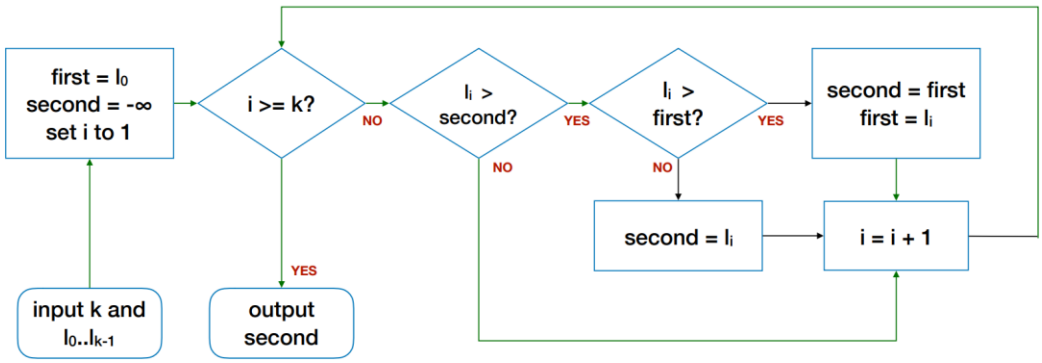
    b. The if-else structure is **incomplete**. (a, b, X, Y are all logical expressions)

```
if (a) {
  // { X };
} else if (b) {
  // { Y };
}
```

    The assertion we can make at // { ?? } is

```
// { ( X || Y) || (!a && !b) }
```

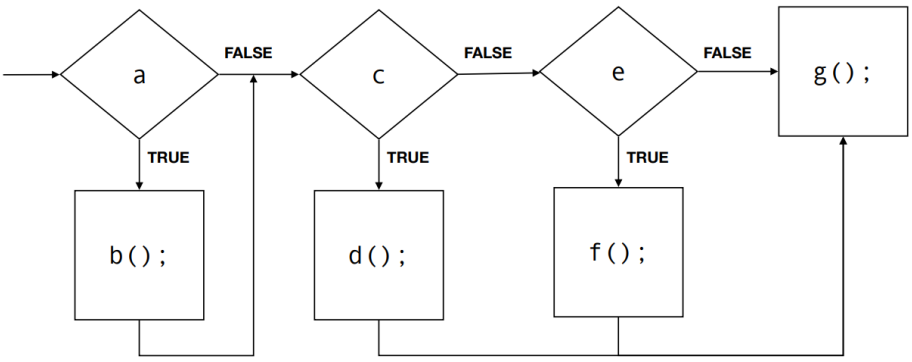
4. Flowchart example to find the **second-large** number in a given list



5. A **loop invariant** is an assertion that is **true before the loop, after each iteration of the loop, and after the loop.**
   And usually, to prove that the **loop invariant** holds **after each iteration of the loop,** we can use wishful thinking aka mathematical induction. So, it will be equal to show that:
    a. it is true at the end of the **first** iteration of the loop

    b. if it is true at the end of the **k-th** iteration of the loop, then it is true at the end of the **(k+1)-th** iteration.
6. **Prove the loop invariant:** if the variables in your **loop invariant** is changed during the execution, change them accordingly in your loop invariant. For example, if our **loop invariant** contains x = … and x is incremented by 1 in the loop body, we should change the loop invariant at this line to be x - 1 =… Do this for all the variables including in the loop invariant. Then once you reach the end of the loop, simplify the expression and see whether it is the same as the loop invariant you have achieved before.
7. Sometimes if it is hard to prove the **loop invariant** from forward, we can use the answer and try to prove it backward (think of what we want at last and then think forwardly about how can we get that)
8. Before we try to form the invariant, do some examples and see what the code is doing!!!
9. To prove a program is false, **one counterexample** is enough. But to prove an algorithm is true, we may use assertion or some other proof techniques. Usually try to find counterexample first!!!
10. If the variables are not too much, always try the **Truth Table** Method since it will make your life and analysis much easier!
11. "*Imprecisions using floating-point values might mean that certain conditions may be evaluated incorrectly*", this occurred when you want to do the **equality test** between a floating number and an integer.
12. When you want to form a single return statement in recursion, **write out the normal if-else recursion first**, then consider under which cases we will get **true** output.
13. When doing questions about **assertion or loop variant**, check **all the options** since there are likely to have some trivial ones answers!!!
14. long x = 2.00 - 1.00; // x will be 1
15. For two integers a and b, as long as abs(b) is bigger than abs(a), a / b = 0.
16. **Redundant Comparisons** are conditions that are always true or always false. Note that this condition can be **part of** the **logical expression**! So, if we can find **any part of the logical expression** that is always true or false, and if **this part is a comparison**, then this part will be a redundant comparison.
17. Using a negative number modulo a positive number, the result is a negative number. i.e. −1234 % 10=−4. We call the % **remainder operator** instead of **module operator**!!! The % operator in C is defined as follows: **x % n** is equivalent to **x - ((x / n) * n**) (where **/** is the integer division operator).
18. Three basic ideas of using wishful thinking to solve a problem recursively:
    a. Assume, by wishful thinking, that we can solve this problem for a smaller n;
    b. Use the solution for smaller n to solve for the original n; and
    c. Solve this problem for the simplest n

19. Create a new **if** block if encountered intersection flowchart like below



20. Loop invariant for the for loop, **the invariant must hold after the initialization and before entering the loop.**