Chapter 3 Recap
==========

Please note that the following are mainly points with some brief explanations to add to the notes you are compiling. The recap is an opportunity to revisit some of the work we have spoken about.

==========
Overview of Understanding Structure
==========
Chapter highlights:
-- The disadvantages of unstructured spaghetti code
-- The three basic structures—sequence, selection, and loop
-- Using a priming input to structure a program
-- The need for structure
-- Recognizing structure
-- Structuring and modularizing unstructured logic

The disadvantages of unstructured spaghetti code
==========
-- Spaghetti code is a term used for any source code that is hard to understand
-- It has no defined structure
-- Factors
- Project requirements are confusing and not clearly understood
- Lack of programming style rules
- Software engineers lack ability and experience
- Developers change and are transferred to new teams - also leads to programmers coding their style in an already existing code base, and unintentionally confusing the current code base
- Development practices are outdated with time and existing systems are not correctly optimized with latest practices.

-- The overuse of one type of code statements leads to unmaintainable programs

-- Consider the following code written using the BASIC programming language - The objective was to write a program to display the numbers from 1 to 100 and the square of the number alongside it

-- **unstructured** version - the reason - while the code can be read and executed as required - the use of the GOTO statement does not follow the rules of programming source code - the code is attempting to repeat instructions, but it would take some time to understand

```
1       i = 0;
2       i = i + 1;
3       PRINT i + " squared = " + (i * i);
4       IF (i >= 100) THEN GOTO 6;
5       GOTO 2;
6       PRINT "Program Completed.";
7       END
```

-- **structured** version - following the rules of structures, the following code's logic is better understood

```
1       FOR i = 1 TO 100
2               PRINT i + " squared = " + (i * i)
3       NEXT i
4       PRINT "Program Completed."
5       END
```

** More reading - **Wikipedia, Spaghetti code**, https://en.wikipedia.org/wiki/Spaghetti_code

-- Basic prevention steps that could and should be taken

- **Comments**
  -- Comments will not only help the programmer, but also another that is reading through the code. Provides clarity where needed.
- **Understanding the Codebase**
  -- When a programmer joins an existing team, it would be good to learn their methods before editing significant areas of coding in software applications
- **Perform Unit Tests**
  --- Performing routine unit tests will help maintain code and reduce the probability spaghetti code
- **Use Light Frameworks**
  --- There are many new frameworks that can be built and used that helps to execute functions in a few lines of code, making your code leaner and helping to fix bugs.
- **Always Double Check**
  --- Always go through your code one more time. Revisiting what you have coded, will validate your source code and its purpose.

The three basic structures—sequence, selection, and loop

==========

-- 3 Logical Structures

-- **Sequential Logic**

- Process instructions one after another in a sequence.

-- **Decision Logic**

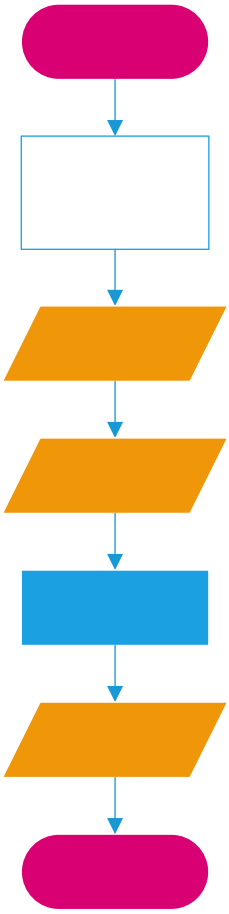
- Select one of two sets of instructions according to the resultant of a condition
- The if..then statement & the if..then/else statement is learnt and implemented in our source code

-- **Loop Logic**

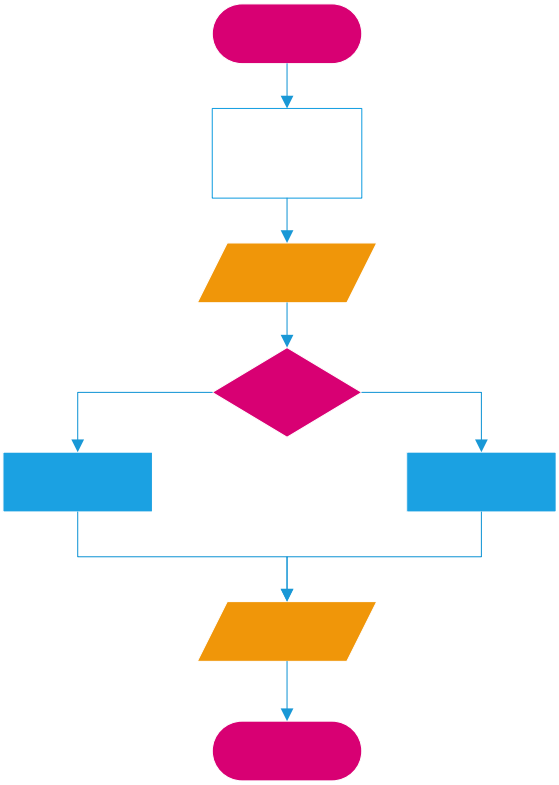
- Enable a program to process the same set of instructions repeatedly
- There are 3 different types of loop statements that will be learnt and implemented
- They are the while..loop, the do..while loop and the for..loop
- In the past, there was the repeat..until loop that is no longer used in programming, but it would be appreciated to still have an understanding of its logic.

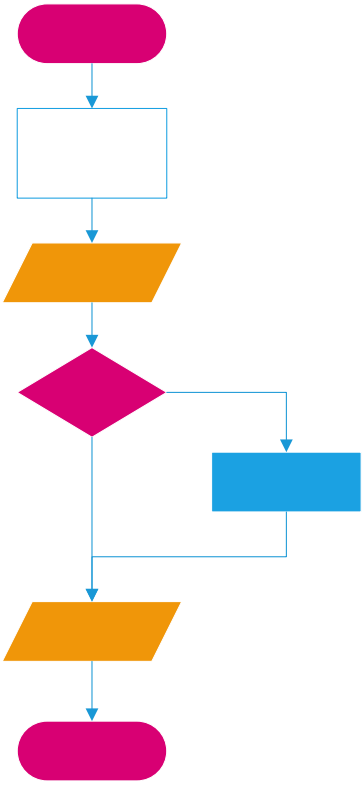
+++ FLOWCHART EXAMPLES

+++ Sequential Logic

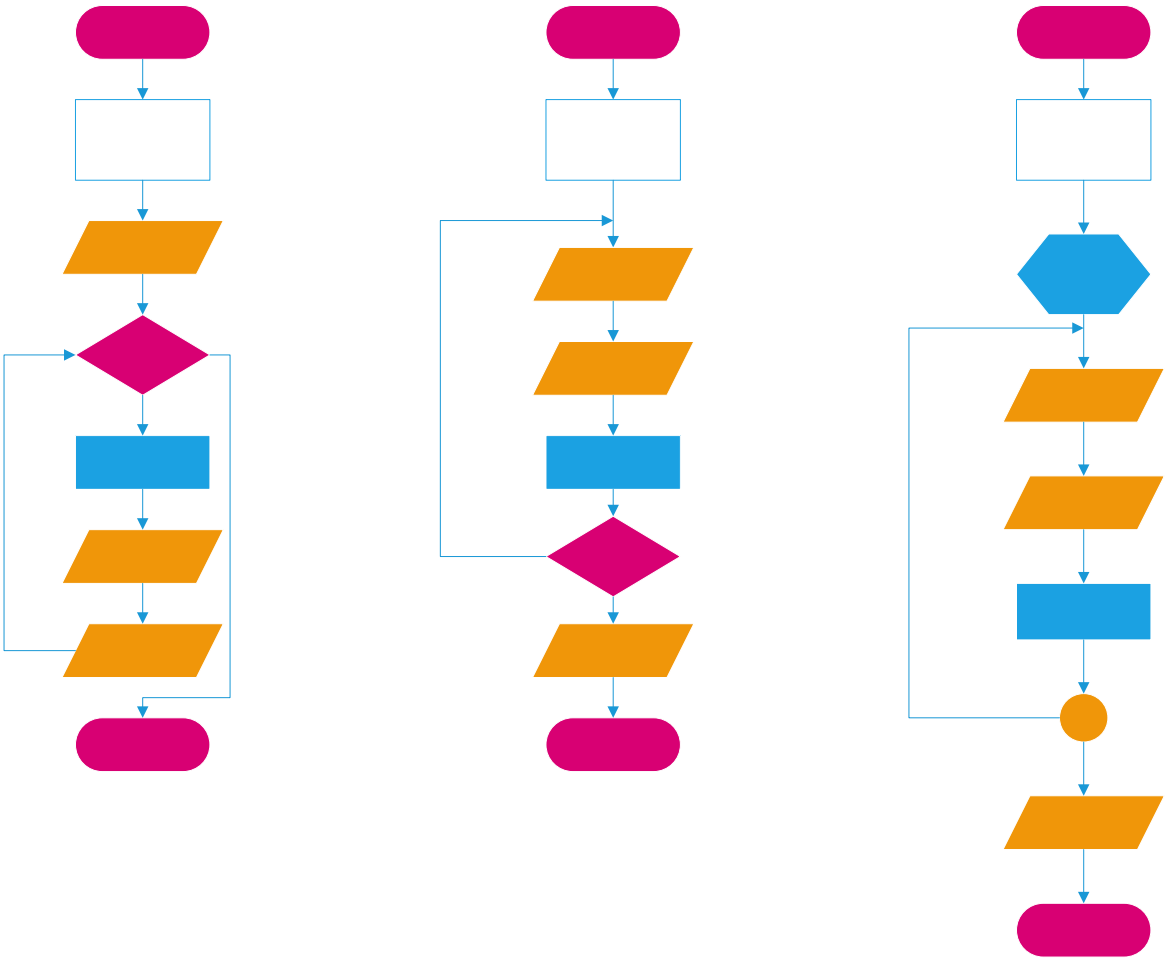


+++ Decision Logic
(*Both flowcharts are variations of the if..then statement*)



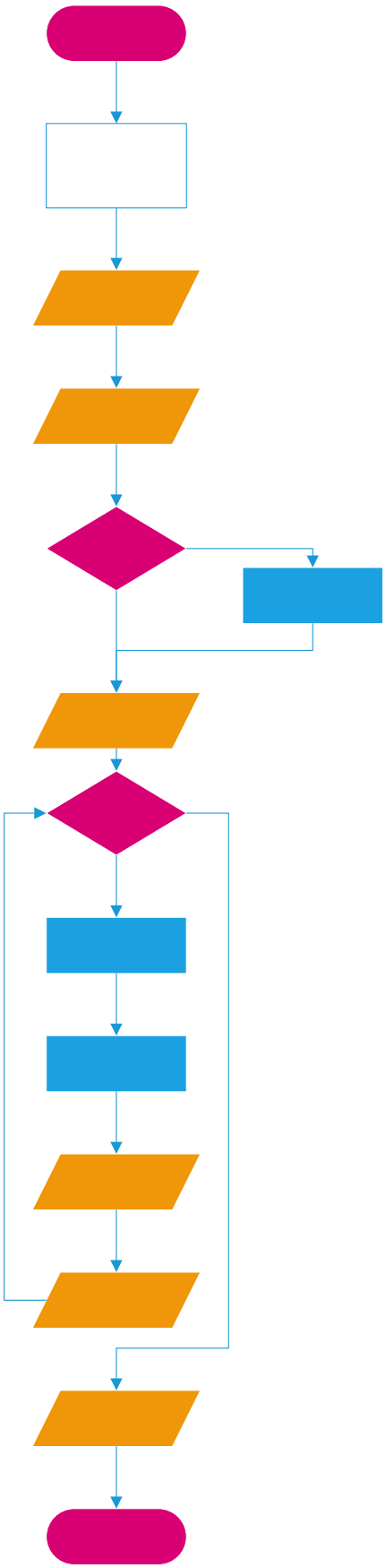


+++ Loop Logic
(*The 3 loops presented is: the while..loop, the do..while loop & the for..loop*)



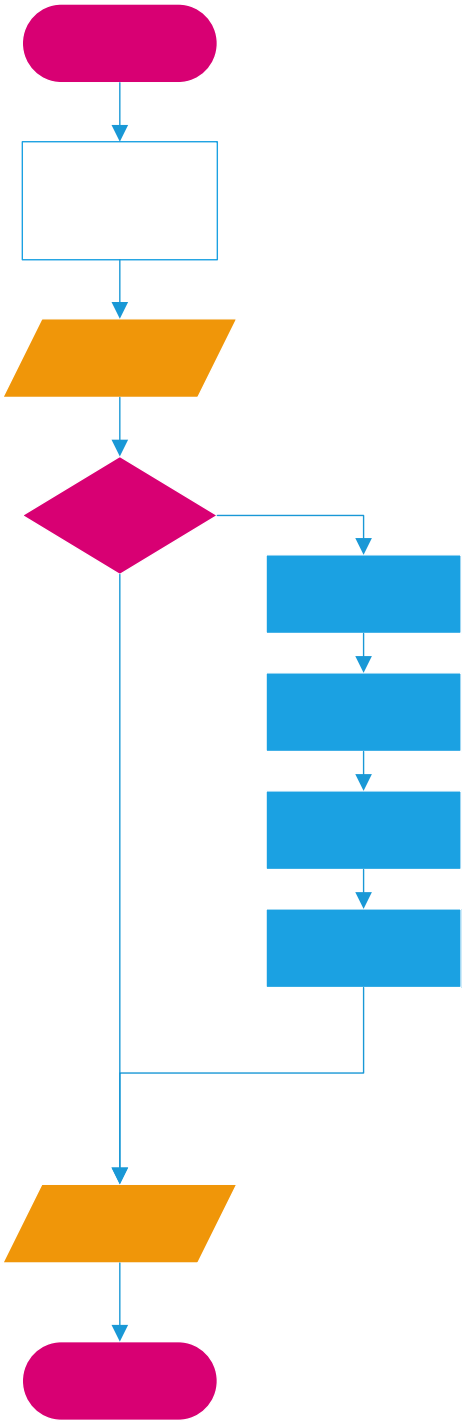
--- Structures can be combined in an infinite number of ways
--- We are allowed to stack logical structures
    -- *For example: sequential logic, followed by decision logic, followed by sequential logic and so on ...*
--- Each of the new statements that we are going to learn will have its own begin & end to recognize in the code what is contained within the statements
--- We are also allowed to nest logical structures
    -- *This gives us an opportunity to place one logic structure inside of another when needed*
--- A group of statements that execute as a single unit is sometimes referred to as a block
    -- This block of code will need to be understood in the future, when there are variables used and possibly declared inside a block, because the scope of the variable comes into question when dealing with code that is outside of a block

+++ Example of Stacked Structures

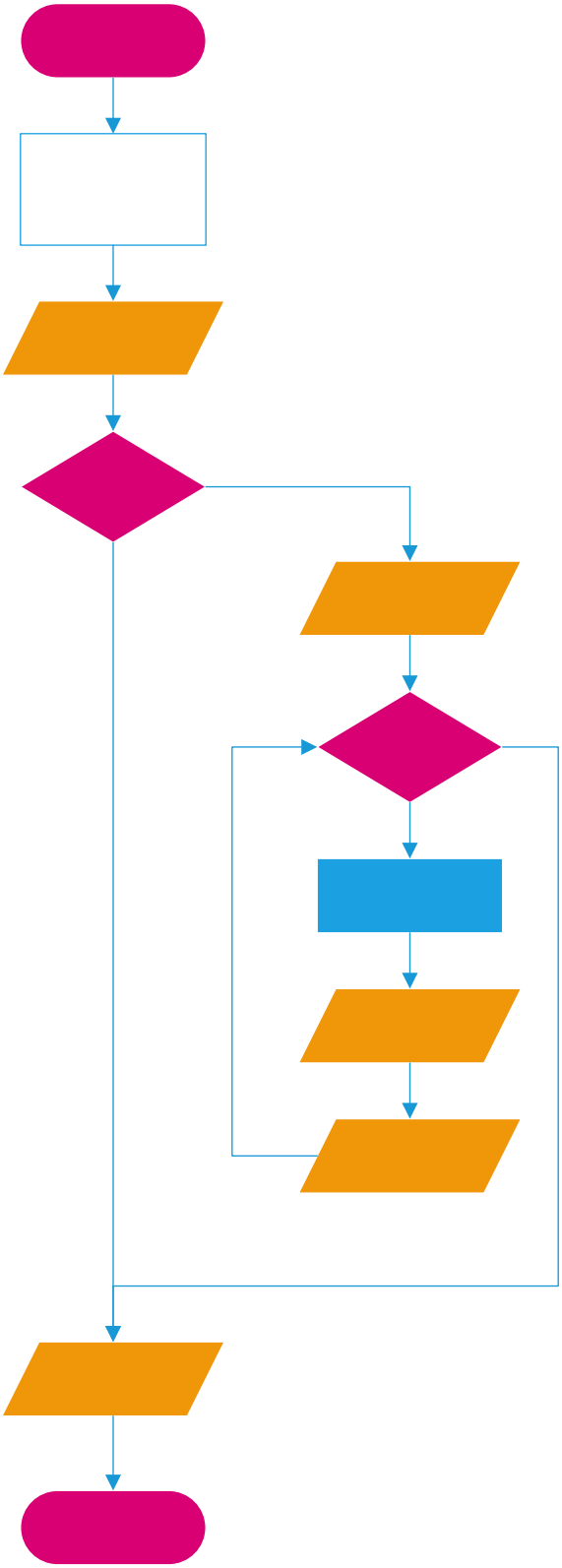


+++ Example of Nested Structures

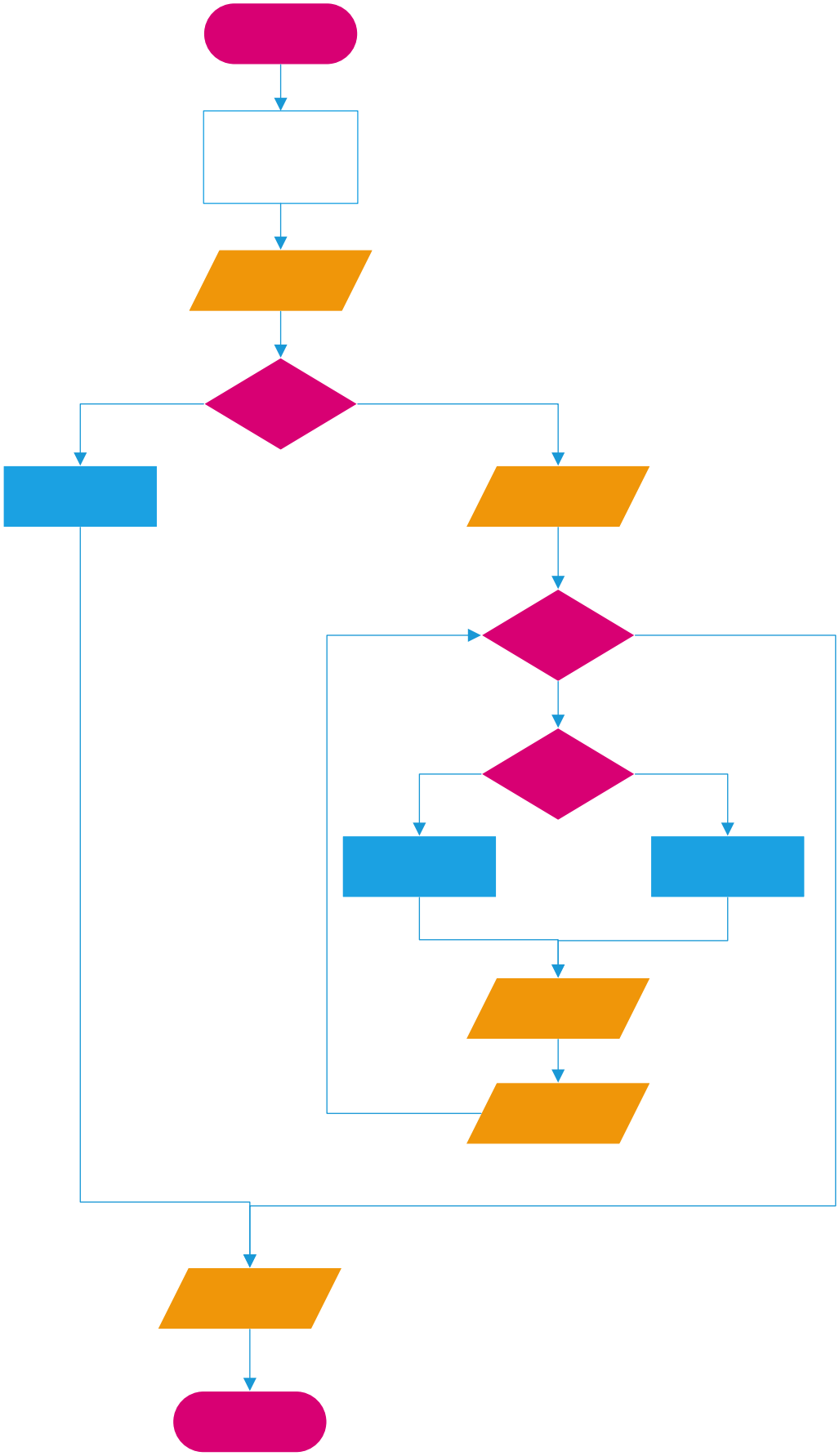
+++ if..then with multiple instructions in true - sequential logic



+++ a loop inside of an if..then statement



+++ an if..then inside a loop, that is inside of an if..then



--- Each structure has a single entry and exit point - this would be noted in the flowchart examples provided

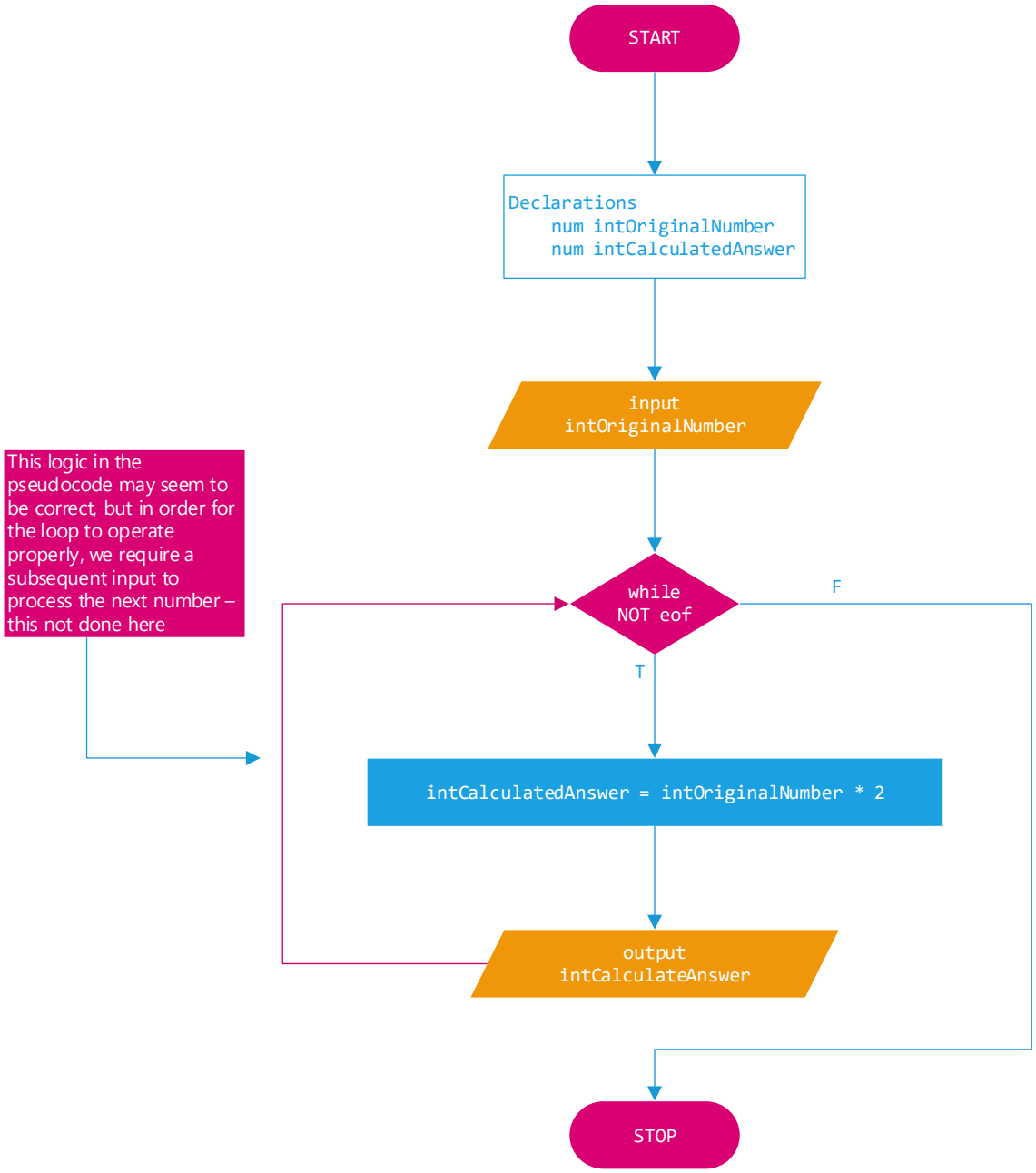
```
================================================================================================================
* The following information is aimed at providing you with the knowledge to understand how to structure
* a program using the logical structures and develop a structured solution
*
* Consider the following points and take note of the information we have learnt thus far:
* 1. Use modules to break the whole into parts, with each part having a particular function.
* 2. Use the three logic structures to develop a solution that flows smoothly from one instruction.
*    to the next. (No jumping from one point of solution to the next)
* 3. Eliminate the rewriting of identical processes by using modules.
* 4. Improve readability, including the logic structures, proper naming of variables and constants,
*    Internal documentation, comments, and proper indentation
*
* Develop your techniques through the learning from others - review programs and best practices to
* understand how they can used and adapted to your solutions
*
================================================================================================================

----------------------------------------------------------------------------------------------------------------
Using a priming input to structure a program
================================================================================================================

* The following information relates to a future lesson we have yet to introduce you to. Since this
chapter is now an overview of the content around the logical structures, it is important to help you
understand what you are going to be learning moving forward.

-- When the time comes to deal with the processing of a collection of records, we will strongly consider
a loop logical structure to process the records. In order to begin the processing, a concept known as
priming input is dealt with at the beginning to allow or deny the program to enter the loop and continue
processing.

-- This idea of dealing with something before entering the loop helps keep the program structured and
also allows us, the programmers, to decide whether the program really needs to enter the loop or not.

-- Another similar term, primer read, can also help with an interaction with the end-user. By asking
them for an initial value, we can decide on the entry into the loop and also decide when to end the
processing of the loop.

-- The following points made help with deciding if a program is structured correctly:
        --- First ask a question
        --- Take action based on the answer
        --- Return to ask the question again
```

-- The following errors are better visualized in flowcharts but please note the pseudocode shared below may not present the mistake just yet.

pseudocode

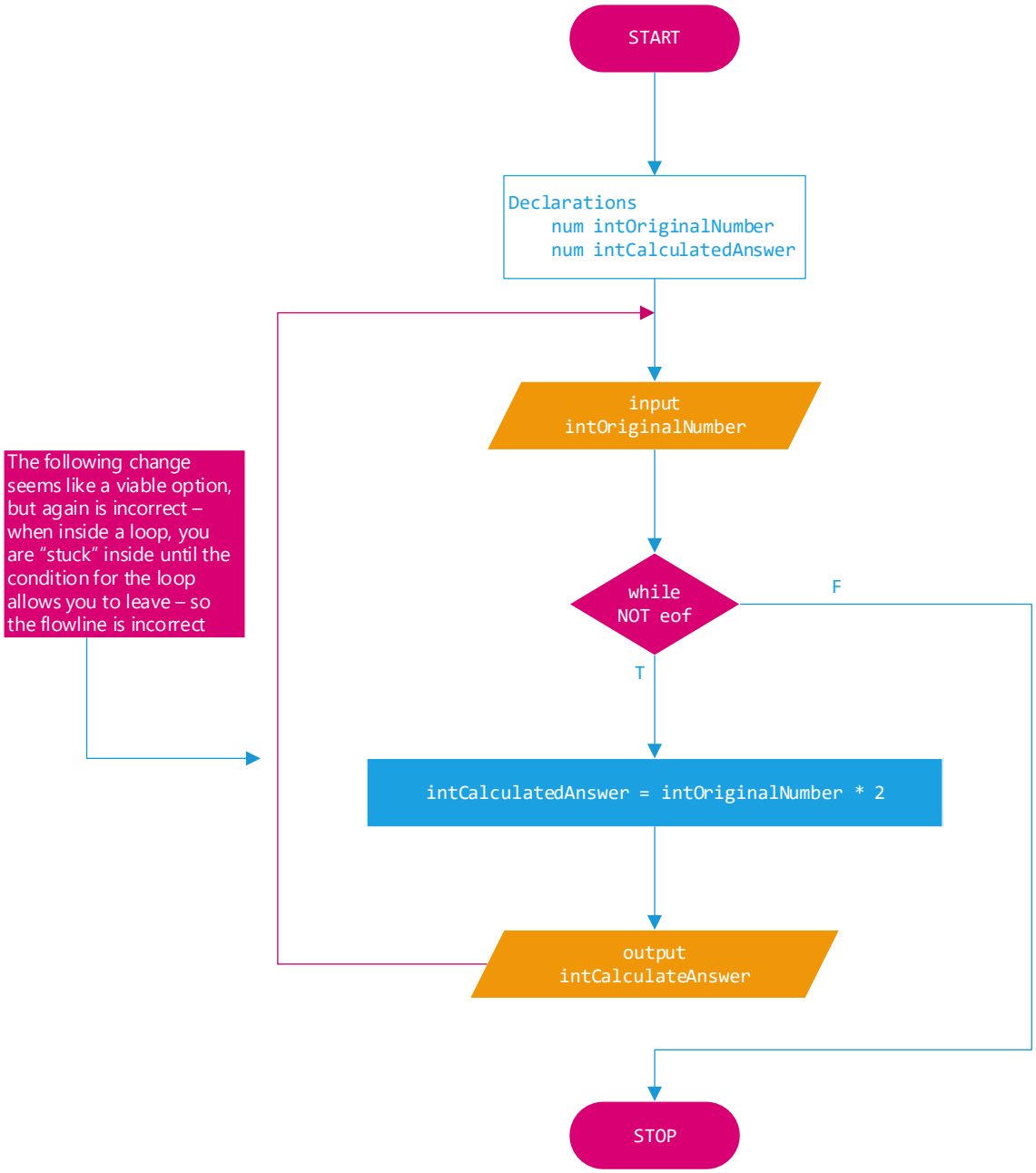
```
0. Start
1. Declarations
        num intOriginalNumber
        num intCalculatedAnswer

2.      input intOriginalNumber
3.      while not eof
                intCalculatedAnswer = intOriginalNumber * 2
                output intCalculatedAnswer
        end while

4. Stop
```

+++ **Flowchart** (Textbook 3-14)



+++ In the code/flowchart above - the loop is dependent on the input of the intOriginalNumber, but it is only executed once. The flowchart showing the flowline for the loop, shows an important piece that the input never is called upon again ... therefore this program will present an infinite loop of sorts and is incorrect.

+++ Flowchart (Textbook 3-15)



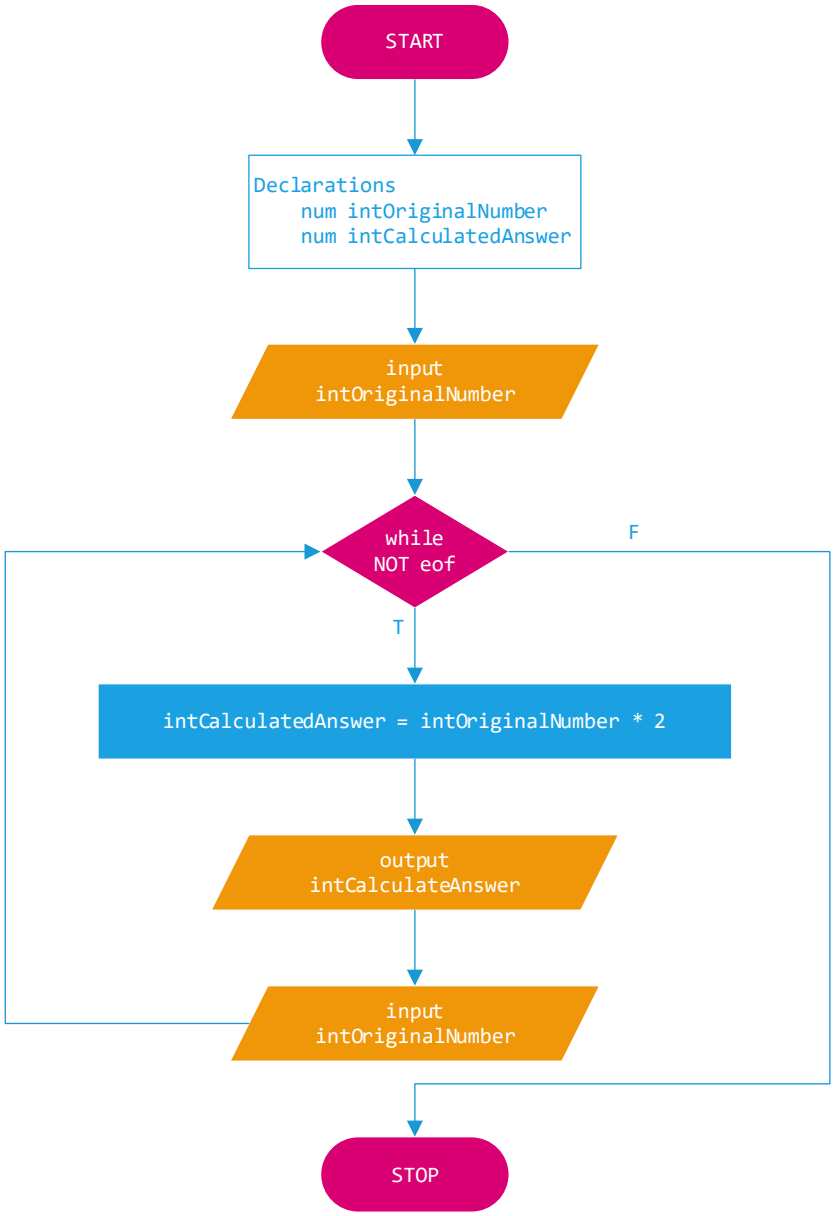
+++ A change is made in the flowchart that shows the flowline flowing to above the input. A change that makes one feel like the above problem was solved. The other program did not request a subsequent input, but this one does. However, in a flowchart and possible pseudocode, this would not be allowed.

+++ The following would be the correct change to have accommodated the first and then subsequent inputs for the program.

pseudocode

```
0. Start
1. Declarations
        num intOriginalNumber
        num intCalculatedAnswer

2.      input intOriginalNumber  // primer read
3.      while not eof
                intCalculatedAnswer = intOriginalNumber * 2
                output intCalculatedAnswer
                input intOriginalNumber // subsequent input
        end while

4. Stop
```

+++ Flowchart



The need for structure

==========

The following reasons for using logical structures within a program will help us better understand the designs of programs.

-- **Clarity**

- Small programs like the one above is simple and easy to understand (and even with the errors, an easy quick fix)

- The larger the program becomes in the future, the more we need to use the logical structures in order to provide a better understanding of our solutions

-- **Professionalism**

- It is just courtesy to consider developing structured programs, since there are going to many programmers who will be reviewing your code for future projects and being able to read through them would be much better when the code is properly structured

-- **Efficiency**

- Many of the present-day programming languages have integrated the use of proper, structured syntax for sequential, decision and loop logical structures.

- Even if there are older programming languages that may not have supported the idea, programmers themselves have made sure that some level of structure is used where needed

-- **Maintenance**

- Aside from you, other developers, within a team and outside, will find it easier to maintain and update code of a structured program

-- **Modularity**

- Since we as programmers have the opportunity to break down a program into a smaller set of procedures and functions (modules), the idea of piecing them back together for the solution as a whole, will end up being a much easier process.

- In future programs, we will also learn how these modules could possibly be used by other programs and therefore for new solutions, the time it takes to code will be cut down since we would be able to use these procedures and functions over and over again.

Recognizing structure

==========

-- It is not only important to code a program, but now at the beginning of your coding journey to also develop the flowcharts to help better understand how the compiler/interpreter is flowing through your code.

-- The flowcharts that are presented here is an opportunity to realise the mistakes that could be made in the drawings and based on the rules we follow, understand what the mistake is and why it is so

+++ Flowchart

Please refer to the Textbook Figure 3-20

When you review the series of flowcharts presented, you are going to realise that in the flowchart on the top left, one of the flowlines is not allowed.

So then, as a series of steps from 1 to 7 – a breakdown of the flowchart is recognized and the error is fixed at **step 5**, the false for the if..then statement is edited, and there is now a duplicate process J added to the flowchart. One of the small things we will begin to learn is that when we are inside a logic structure, we are inside and cannot leave or “jump” out of it, just because a flowline looks like it can work.