[0405] Structured Programming

Back in the 1960s when Structured Programming was invented, computers were still relatively new and primitive, but they were evolving quickly. They had reached a point where the hardware was fast enough to do "interesting" things for businesses and businesses were trying to apply the number-crunching capabilities of these new mainframe computers to tasks traditionally done by humans.

Soon, however, a big problem started to become apparent. The problem was that the hardware was no longer the bottleneck when it came to using computers. The bottleneck had become the ever-increasing complexity of the software. Programs had increased in size and complexity to the point where no one person could understand everything that was going on inside of them. The ad-hoc approaches to programming that had been used up until that time were resulting is what is commonly called "spaghetti code." Code with branches and loops and variable changes that were almost impossible to understand and maintain. It was quickly becoming painfully apparent that a new approach was needed. Enter Edsgar Dijkstra and the Structure Programming "revolution."

Structured Programming introduces three key concepts:

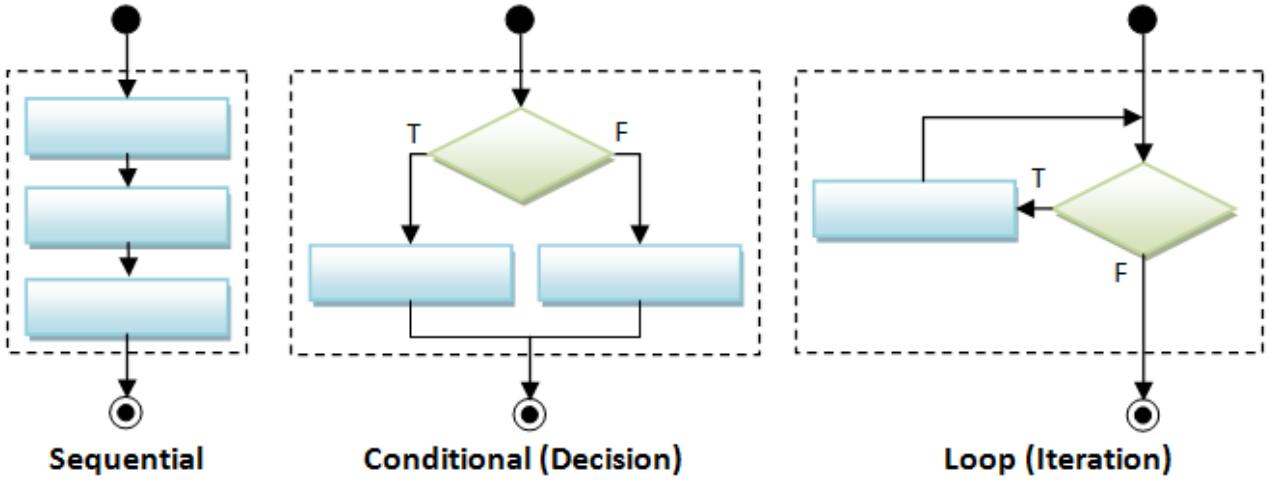
- Limited Control Structures
- Scoped Variables
- Blocks of Functionality

Let's look at each of these separately...

Limited Control Structures

In Structured Programming, all programs are composed of just three control structures:

- Sequence - the normal sequence of program statements
- Selection - "if" or "if/else" statements
- Iteration - "do/while" or "while/do"



Over the years, a couple of other control statements have been adopted such as "switch" and "for/next", but those are just convenient variations on the initial program statements.

Note that GOTO is specifically not allowed here. The ability to arbitrarily jump from one section of code to another was (and still is) considered to be too disruptive to the orderly, understandable flow of the program. (Structured Programming purist banned GOTO altogether, however it has made a small, limited return when it comes to error handling as we will see later.)

Blocks of Functionality

The most fundamental characteristic of structured code is that it is composed of properly nested code segments that are entered only at the top and exited only from the bottom. They are "single-entry, single-exit blocks."

In addition, the most important requirement for any program unit is that it addresses only a single logically coherent task - i.e., ***it should do one thing and do it well***. It needs to be "self-contained and independent of other modules in the system." The way you achieve this independence is by:

- Avoiding the unnecessary modification of global variables
- Declaring all temporary variables as local to the module
- Avoiding changes to input variables that were passed by reference
  - Corollary: Only pass input variables by value
- Only do what you are supposed to do based on the specification of the problem (and your module's name)

Scoped Variables

The term "scope" refers to the places inside of a program where a variable is "visible" to the code. i.e., Where can the code use the variable? When computers were first invented, all variables were "global" in scope - meaning that any part of a program could read from them and (more problematically) any part of a program could change them. It also meant that programmers couldn't reuse variable names - "x" was "x" everywhere in the program! If two different programmers wrote different modules that modified "x", they probably created a bug because they were unknowingly modifying each other's variable. What a mess!

The solution to this problem was to invent "local" scope and eliminate (or at least greatly reduce) the use of global variables. A local variable is only visible from within the "scope" (i.e., the block of code) that it was created in. The variable "x" that was created in one block of code was completely different from another variable called "x" that was created in a different block of code. Not only did those two "x's" refer to a different memory locations, they also could only be seen and used by code inside the same block they were created in.

Code Reuse and Code Sharing

The concept of Scoping, when combined with the concept of Functional Blocks of code and Structured Programming in general, opened the door for programmers to start creating reusable, sharable libraries of code. Instead of writing everything from scratch each time, programmers could now use code that other people had written to create their programs. Unfortunately, while code sharing and reuse was now possible, it wasn't easy. There was still one KEY piece of the puzzle that was missing. We'll talk more about that missing piece in a couple of chapters...