CS 115 Lecture

Structured Programming

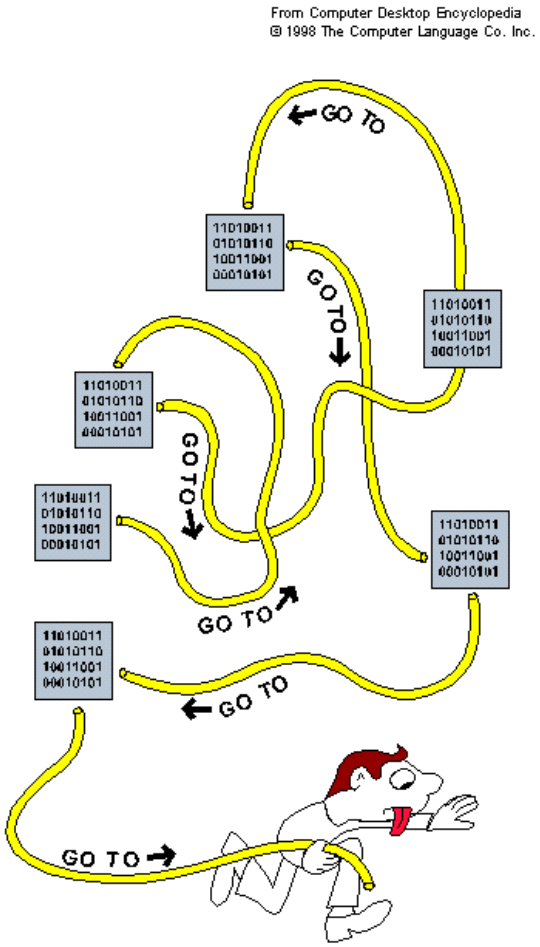
Taken from notes by Dr. Neil Moore

The bad old days: GOTO

In the early days of programming, we didn't have for loops, if statements, etc.

- Instead, we had just "if this is true, go to 10".
- You could use that to skip over code (like an **if** does)
- … or go back to an earlier line to make a loop
- This was very tedious and error prone
  - Especially if something had to be changed
  - "Spaghetti code": trying to trace a program's execution was like trying to trace one strand of spaghetti in a plate of it

Spaghetti code



Structured programming

- In the 1960's, computer scientists started to think about how to write programs that were easier to understand and modify.
  - Edsger Dijkstra, "Go To Statement Considered Harmful" (1968)
- They introduced the paradigm of **structured programming**.
  - Patterns that lead to easier-to-understand code
    - Easier to test and debug
    - Easier to modify and maintain
    - Easier to collaborate on large programs

Data structures and Control structures

- We've already seen a little about **data structures**
  - The ways of organizing data in a program
  - Simple ones: constants and variables
  - More complex: graphics objects, strings, lists, …
- **Control structures** are ways of controlling the execution of a program
  - which statements execute, and in which order

The three basic control structures

In 1966, Böhm and Jacopini showed that any program using “go to” could be rearranged to use only three simple control structures

- Sequence
- Selection
- Iteration
- Added a fourth later: **Subprograms** (more in chapter 5)

Each of these control structures has two important guarantees:

- Only one way to **enter** the control structure
- Only one way to **exit** the control structure
- *One entrance, one exit.*

The sequence control structure

"Sequencing" or "sequential execution" just means:

running one statement after another

- In Python we write one line after the next
- "The default" control structure
- Guarantees unique to sequence
  - The steps will execute in the order given
  - Steps will not be skipped
  - It will always start at the first statement …
  - And finish at the last statement of the sequence

Selection control structure

"Selection" means choosing which code to run based on a condition or question

- In Python, an **if-else** statement
- Two branches, based on True and False
  - Each branch is another control structure (most often a sequence, can be a loop or another if)
- Guarantees:
  - Always starts with the question/condition
  - Runs one branch or the other, never both
  - … and never neither – MUST do one of the two
- Avoid **dead code**: code that is never executed – not just not executed on one particular run, but NEVER executed
  - Usually because the condition is always False

Iteration control structure

"Iteration" means running code multiple times (a loop)

- In structured programming, "repeat this body until a condition is false"
- In Python, a **while** loop (in about a week)
  - **for** loops are a special case of iteration
- Guarantees:
  - Always starts with the question/condition
  - If the condition is True, executes the **entire** body, then comes back to the condition
  - If the condition is False, leaves the loop
- Beware of **infinite loops** where the condition is always True

Subprogram control structure

Sometimes we need to repeat the same code in several different places

- It would be nice if we didn't have to write the code multiple times
- A **subprogram** is a chunk of the code treated as a single unit
- When we need to execute that code, we **call (invoke)** the subprogram
  - The call runs the subprogram, waits for it to finish
  - Then execution keeps going from where the call is
  - Sometimes we send values to the subprogram
  - Sometimes the subprogram sends value(s) back

Subprogram control structure

- In Python, subprograms are called **functions**
  - **Arguments** are the values we send to the function
  - And the function can **return** a result (or results)
  - Can you think of Python functions that
    - Take one or more arguments?
    - Take no arguments?
    - Return a result?
    - Don’t return a result?

Subprogram control structure

- Guarantees for subprograms:
  - Forces the invoking code to pause execution
  - Starts execution at top of subprogram code
  - Completes execution at bottom of subprogram
  - Always returns execution control to the point where the subprogram was invoked
  - Does NOT execute unless invoked (called)

Control structures summary

- **Sequence** (one statement after another, easy to forget its name!)
- **Selection** (conditionals: `if` and variations)
- **Iteration** (loops: `for` and `while`)
- **Subprogram** (functions: `def` and calls)