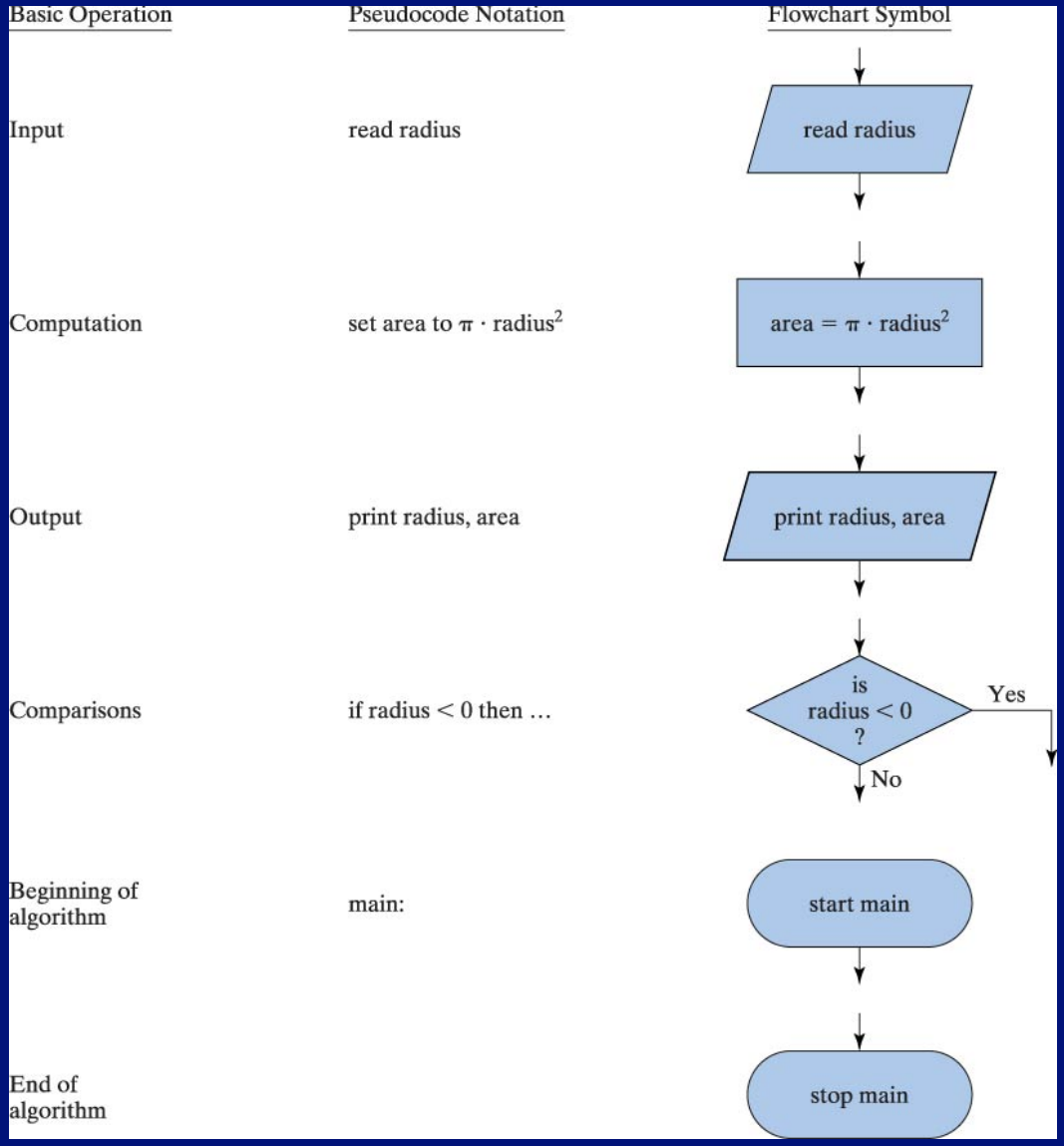
Structured programming summary & intro modularity

Recap flow of control (Ch. 2&3)
Introduce functions (Ch. 4&5)

Notation for algorithms

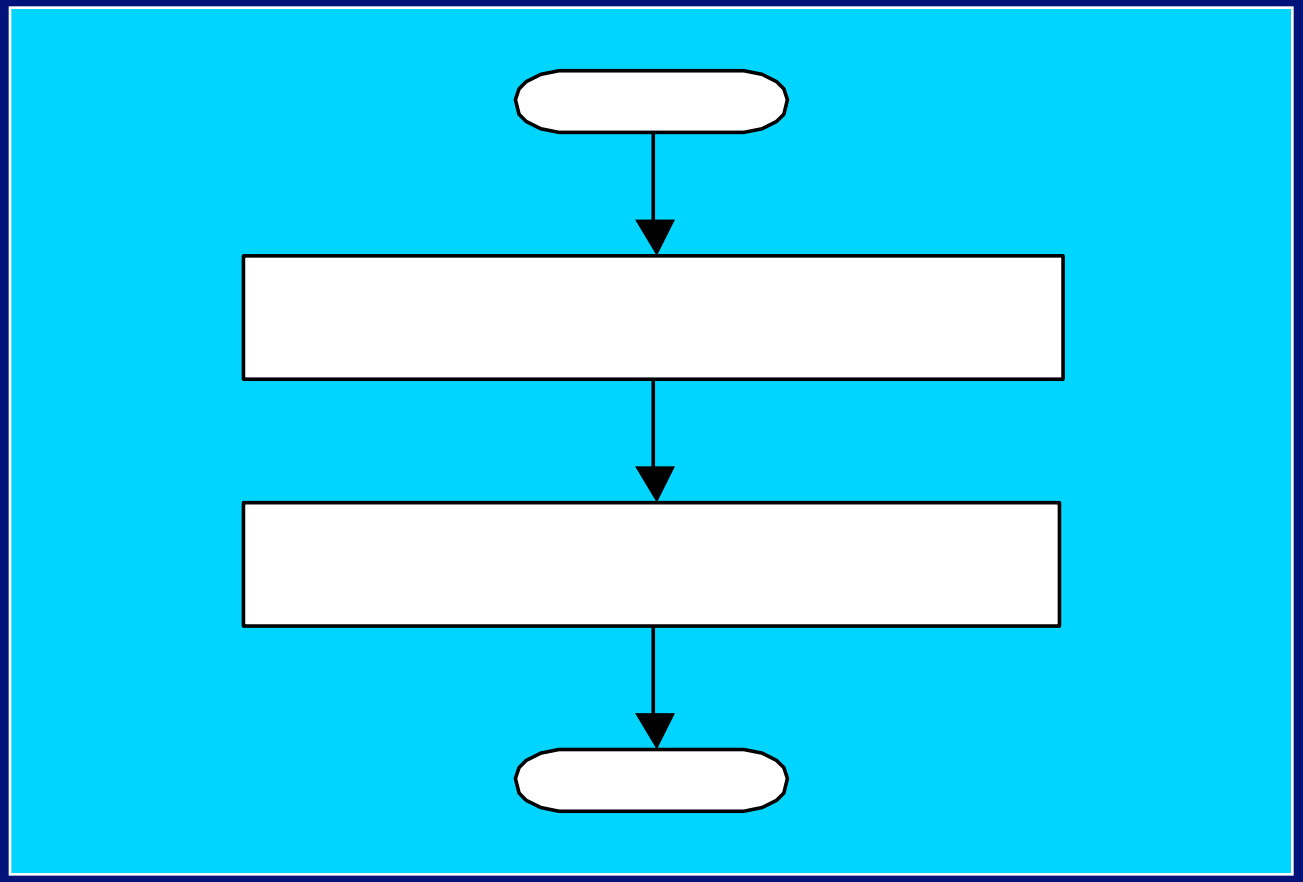
- Example pseudocode notation (not a "standard"), and flowchart symbols (relatively standard)



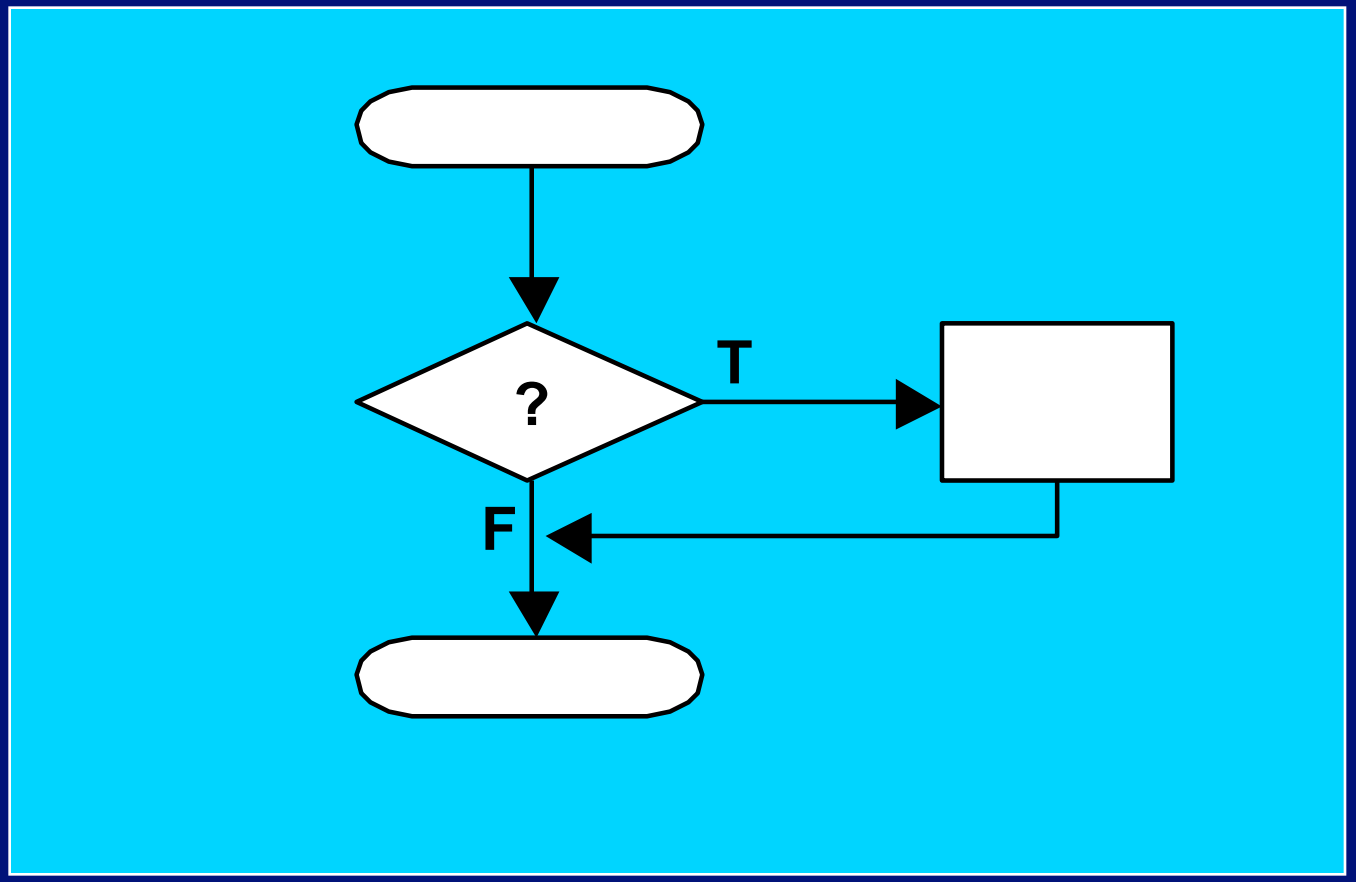
C++'s 7 basic control structures

- 1st is trivial: sequence structure
- 3 choices of selection structures:
  - `if`
  - `if/else`
  - `switch`
- 3 choices of repetition structures:
  - `while`
  - `for`
  - `do/while`

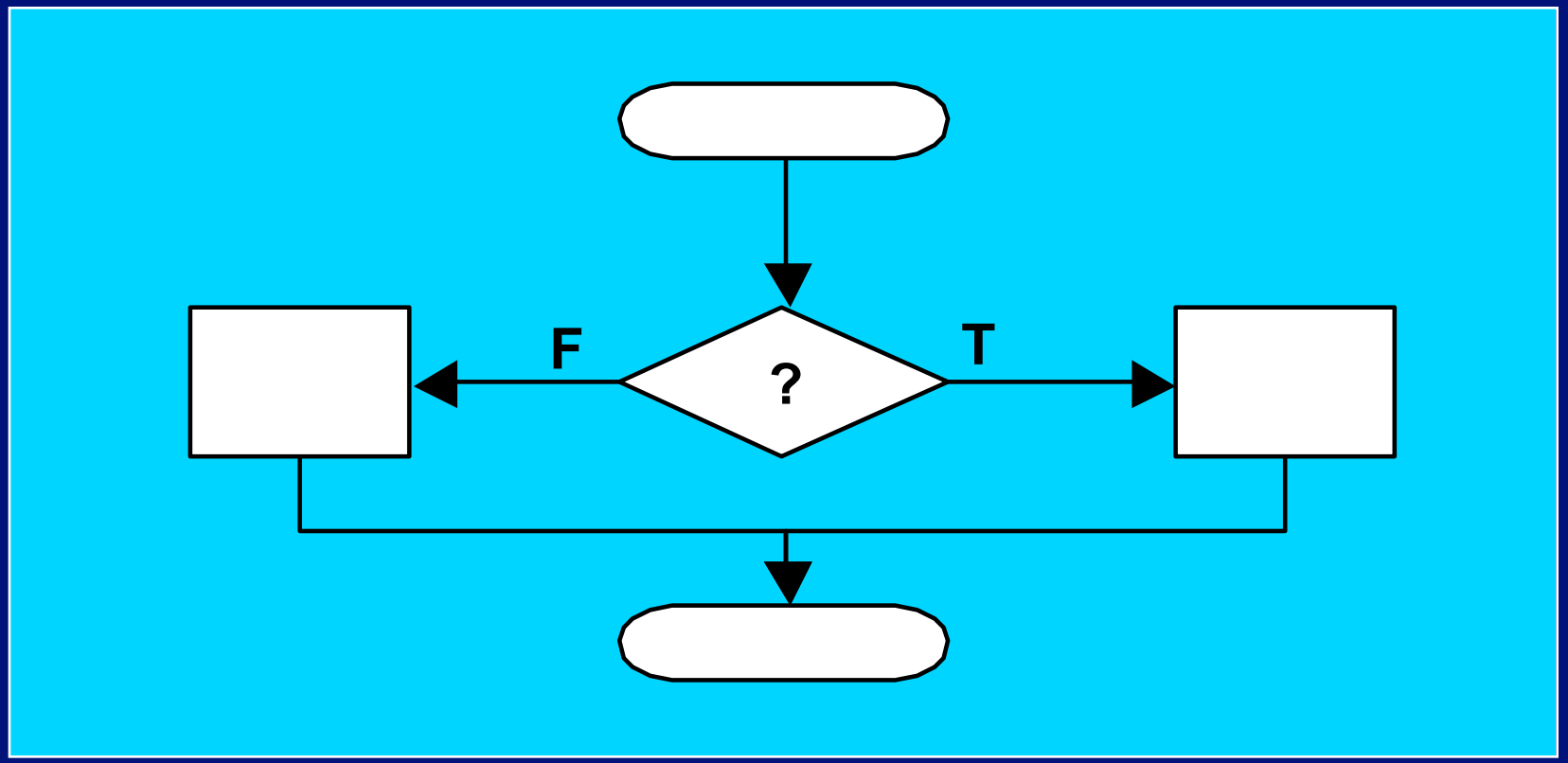
Sequence (it really is a structure)



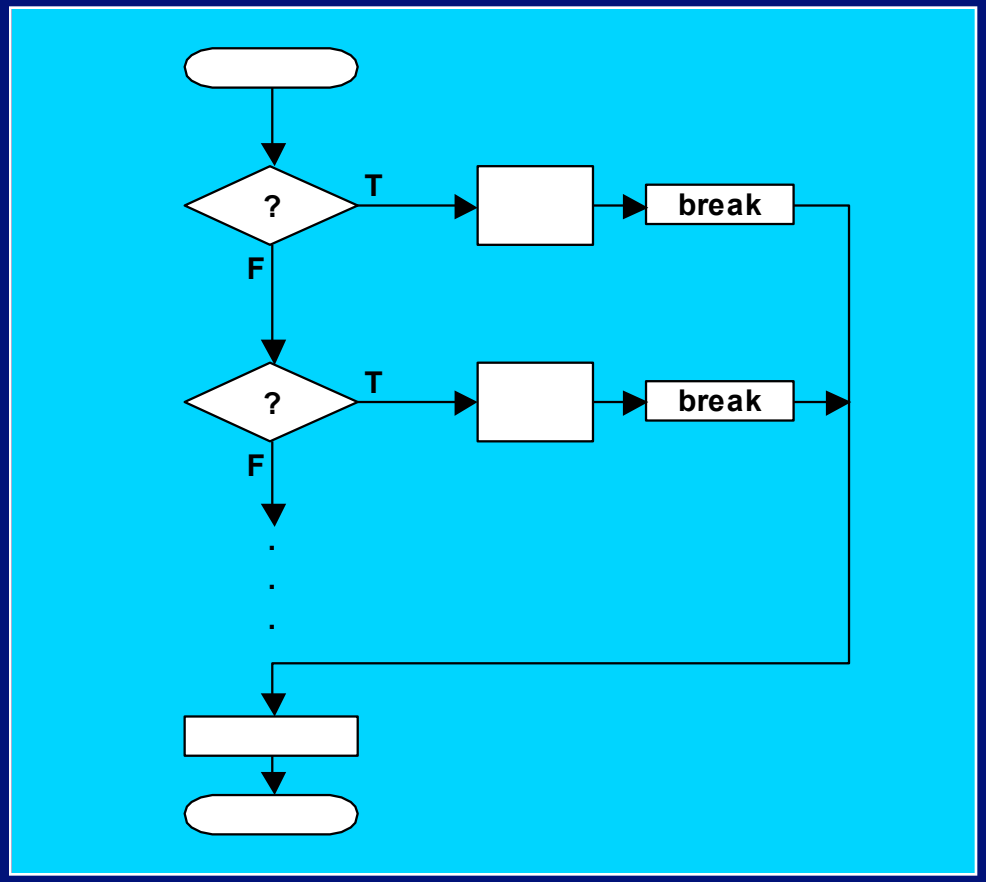
if Selection Structure



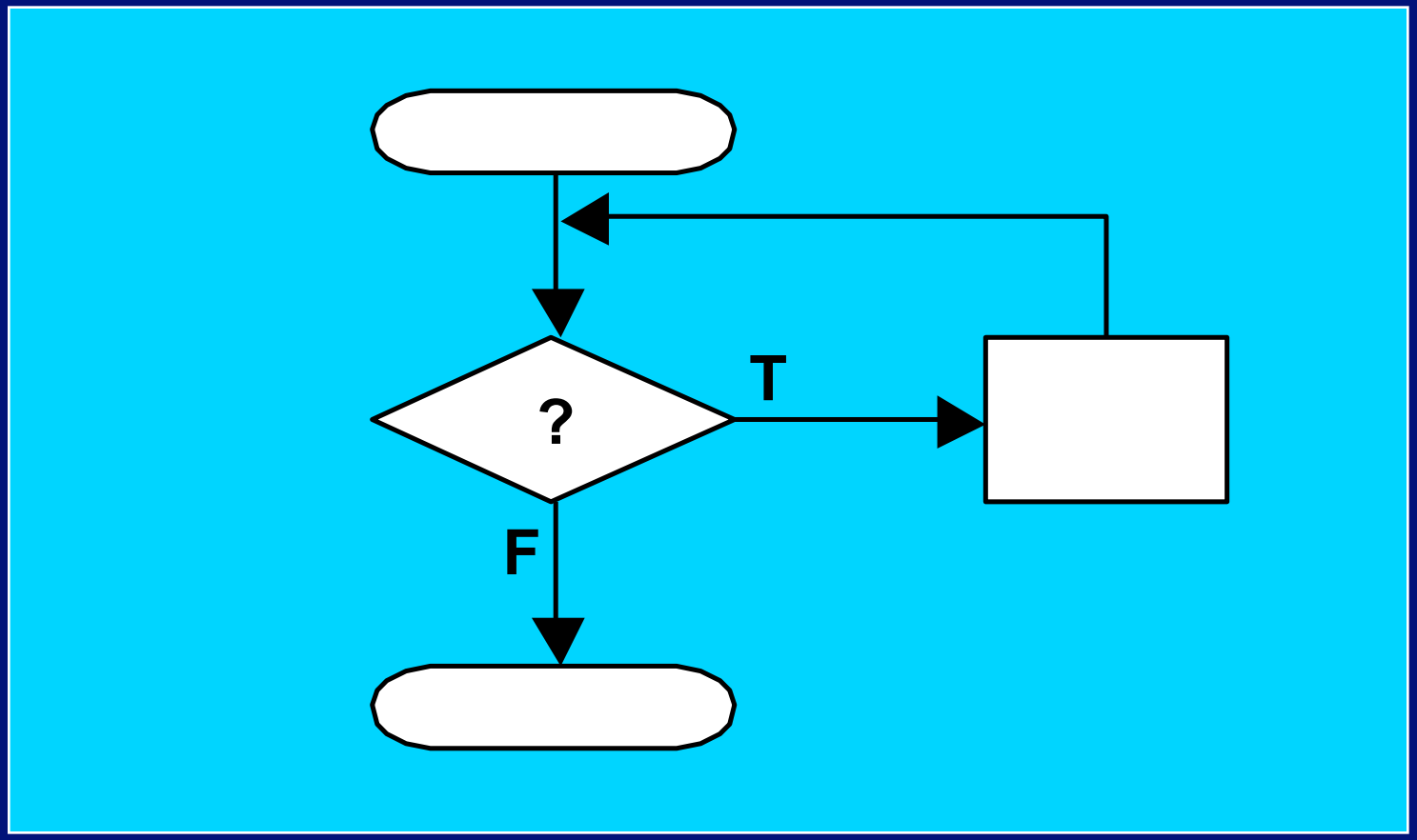
if/else Selection Structure



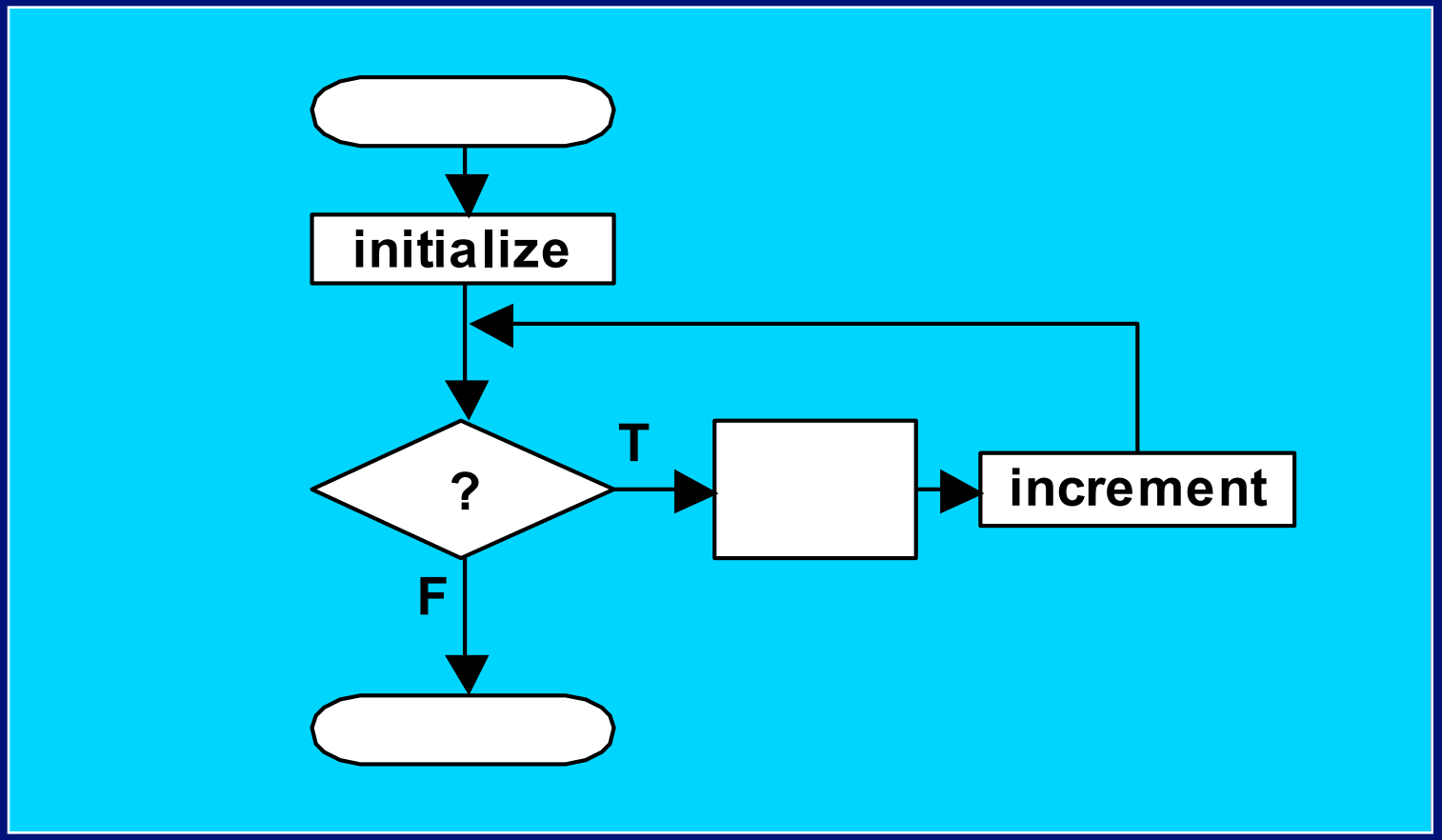
switch Selection Structure



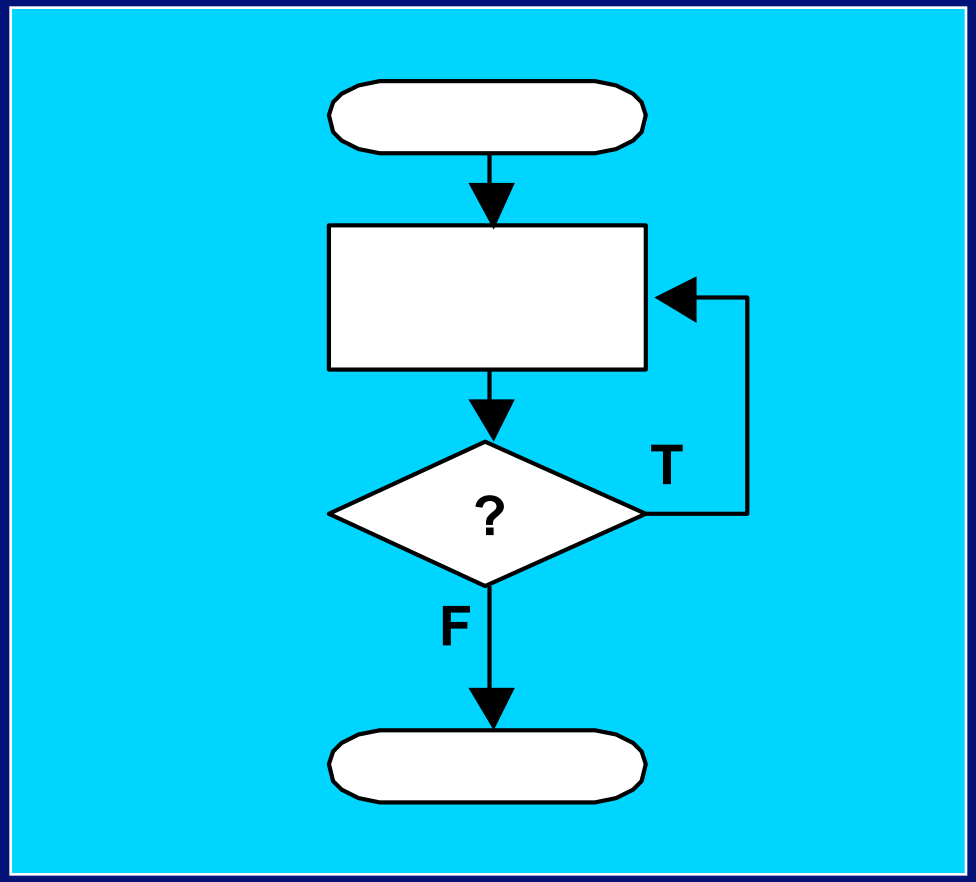
while Iteration Structure



for Iteration Structure

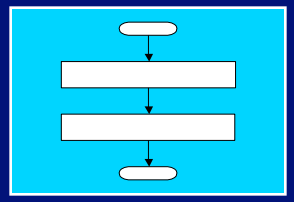


do/while Iteration Structure

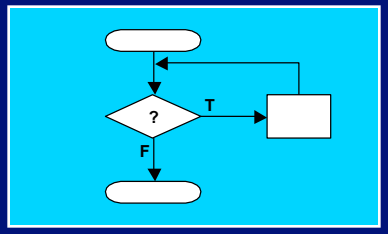


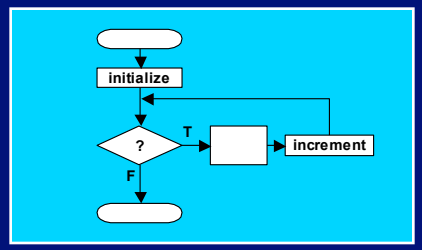
Notice rectangles in every one

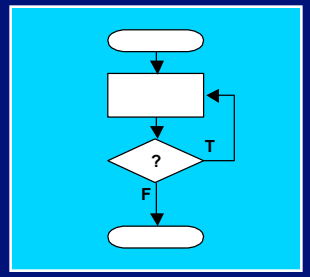
Sequence



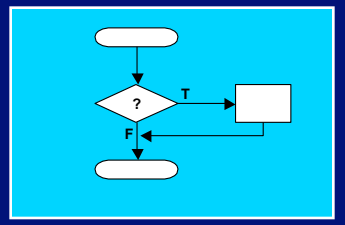
Iteration

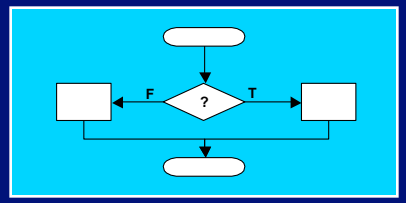


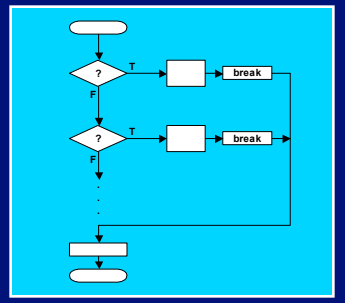




Selection







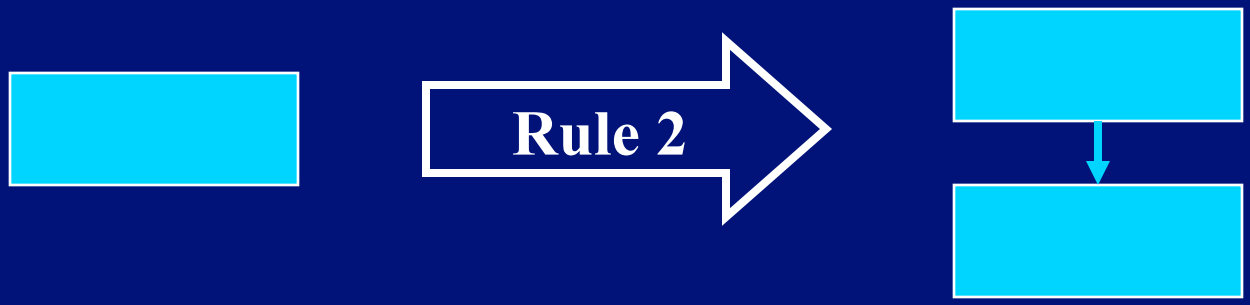
Structure "rule" #1: start with the simplest flowchart

- One rectangle
- A good (and widely applicable) example:

  get some data, calculate and show some results

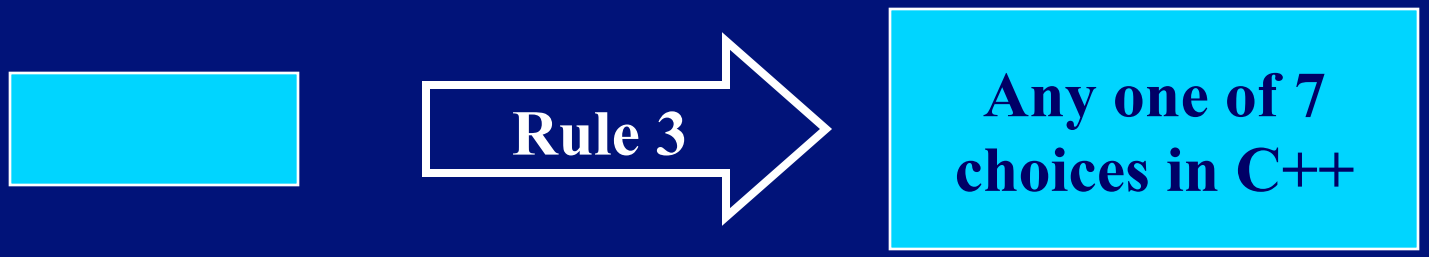
- Really just a way to start; clarifies the "big picture"

Rule #2: replace any rectangle by two rectangles in sequence



- This "stacking rule" can apply repeatedly: one→two, two→three, …
  For example:
  1. Get data
  2. Process
  3. Show results

Rule #3: replace any rectangle by any control structure



- This "nesting rule" also applies repeatedly, as each control structure has rectangles
- e.g., nest a `while` loop in an `if` structure:

```
if  (n > 0)
    while (i < n)
        cout << i++;
```

Rule #4: apply rules #2 and #3 repeatedly, and in any order

- Stack, nest, stack, nest, nest, stack, … gets more and more detailed as one proceeds
  - Think of control structures as building blocks that can be *combined in two ways only*.
  - Captures the essence of stepwise refinement: keep adding details as they arise
    - And keep adding control structures as long as more are needed

Modularity – another structured programming idea

- Function = the simplest type of C++ module
- Idea: let modules solve problem *parts* – then combine the parts to solve whole problems
  - Abstraction benefits – *details are hidden* in a module to reduce complexity of overall solution
  - Reusability benefits – maybe use it many times
  - Benefits of unit tests – be confident each of the parts work properly

Using functions to solve problems

- Think: you might be able to directly translate an algorithm into a series of function calls

```
mydata = getData();
results = process(mydata);
showResults(results);
```

- In turn, the function `process()` might do:

```
intermediateResult = calculate(x, y);
```

  where `calculate` is another function, to perform a difficult calculation involving `x` and `y`.

  – … "top-down programming by stepwise refinement"