

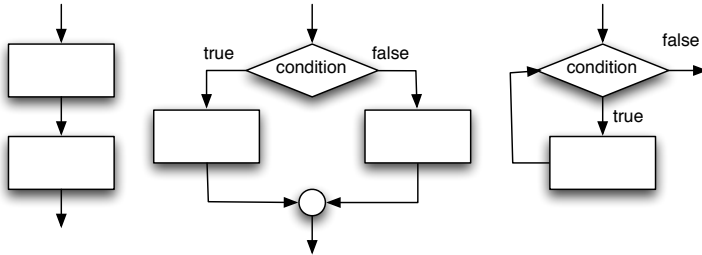
## Structure of Programming Languages – Lecture 4

CS 636 – 536

April 27, 2009

- 1 Minimal Control
  - Boehm and Jacopini, 1966
- 2 Unstructured Control
  - What's Wrong With GoTo?
- 3 Introduction to Lisp
  - Functional Programming
- 4 Structured Control
  - Conditionals p.260

# One-in : One-out



## Sequence, If..else, and While Suffice

- Jacopini: Any flow diagram can be written, without arbitrary GOTOs, in terms of conditionals, while loops, and sequences
- Edsger Dijkstra– 1965, 1968: GOTO Statement Considered Harmful.  
Programs using only these three control structures can be written more rapidly and with less likelihood of errors.
- Rubin and others, 1968: GOTO Considered Harmful  
Considered Harmful

## Conditional and Recursion Also Suffice

Sequences, conditionals, and while are not the only possible basis for a programming language. Full power is achieved by only two things:

- A conditional statement.
- Functions with parameters and recursive function calls.

Sequences and loops are not needed.

## GoTo: Bad Effects on Compilation

- The global symbol table becomes cluttered if many labels are used.
- The location of any forward branch must be remembered. Later, when any label is found, the list of incomplete forward branches must be searched and any branch that refers to that label must be patched up and deleted from the list. In a program with a lot of labels, this is slow and inefficient.

## GoTo: Bad Effects on Proofs of Correctness

- Correctness proofs start by breaking the program into one-in/one-out sections and proving that whenever some correctness property is true at the beginning of a section, it is also true at the end.
- Such proofs become much easier when the size of each section is small.
- But a program cannot be divided between a GOTO and its target label.

Thus the more that GOTOs are used, and the more tangled they become, the harder it is to prove that the program does its intended job.

## GoTo: Bad Effects on Proofs of Correctness

- Some of the GOTOs might be data-dependent conditional branches.
- With these, the flow of control cannot be predicted by looking at the program, and the number of possible control paths that must be checked is doubled by each IF.
- With many branches we get a combinatorial explosion of possible control paths, and complete debugging of a long program becomes highly unlikely.

## GoTo: Bad Human Engineering Properties

- Poor visual correspondence between the true structure of the GOTO program and its apparent structure, represented by the order of lines on the page.
- The end of such a program tends to be a series of stray clauses from a bunch of unrelated IF's. Things that belong together are not placed together in the source code, and the code has poor lexical coherence.
- The programmer will often write an IF and simply forget to write the code for the remote section.
- Debugging such code is more difficult because it requires lots of paging back and forth.

Eventually the flow of control can become simply too confusing to follow.

# Spaghetti Code Without GOTO

To build GOTO-free spaghetti code,

- Define several boolean flag variables.
- Have long, highly nested loops and if-statements.
- Change one or more boolean flags in several places
- Do not supply a state-transition diagram.

# LISP, Scheme, and Functional Languages

- The Functional idea: no sequences, no assignment.
- LISP: early 1960's: List processing language. Small and simple, but powerful
- Scheme: A lexically-scoped LISP
- Common Lisp: A collection of all the parts people have ever attached to a LISP system.
- Miranda, ML, Haskell: Updated in both syntax (all those parentheses) and semantics (tuples).

# Introduction to LISP

- How do we control sequencing?
- How do we achieve repetition?
- Why don't we need assignment?

Please ask questions about LISP.

# LISP gcd

```
; ----- The gcd function in LISP.
(defun mygcd(x y)
  (cond ((or (< x 0) (< y 0)) (gcd (abs x) (abs y)) )
        ((< x y)              (gcd y x)              )
        ((= y 0)              x                      )
        (T                    (gcd y (rem x y))      ) ))
```

```
; ----- Calling gcd
(defun gcdprint(a b)
  (let (
    (f (list "The gcd of " a " and " b " is " (mygcd a b)))
    )
    (mapcar 'princ f)
    (terpri)
  )
)
```

# Conditionals

- IF ... GO ...
- IF ... break
- IF ... THEN ... ELSE ...
- IF ... ELSEIF ... ELSE ...
- SWITCH or CASE
- ... ? ... : ... ; conditional expression
- (cond ... )