Algol Part 2

CS4100

March 4, 2011

Based on slides by Istvan Jonyer

Algol's Objectives

- The language should be very close to mathematical notation
- Should be useful in publications to describe algorithms
- Mechanically translatable to machine code

Structural Organization

- Hierarchically structured language
  - Nesting is introduced (Fortran did not use nesting)
  - Control structures can also be nested
    - One can be made the body of the other

```
if N > 0 then
  for i := 1 step 1 until N do
       sum := sum + Data[i]
```

    - Advantage: decreases the number of GOTOs required
- Reserved words

Constructs

- Declarative or Imperative
  - (like in FORTRAN)
- Variable declarations
  - Types: integer, real, Boolean

```
integer i, j, k
```

  - Lower bounds of arrays need not be 1

```
real array Data[-50:50]
```

  - Switch, like FORTRAN's computed GOTO
- Subprogram declarations
  - Keyword: **procedure** and
  - Procedures can be *typed* (functions) and *untyped*

```
real procedure dist(x1, y1, x2, y2);
  real x1, y1, x2, y2;
  dist = sqrt((x1 - x2)^2 + (y1 - y2)^2)
```

Imperative Constructs

- Computational
  - – Assignment: “variable := expression”
  - – Operators:
    - Arithmetic: +, -, *, etc.
    - Relational: =, <, >, ≥, etc.
    - Logic: ∧, ∨, ¬, etc.
  - – Why is assignment ‘:=’ and not ‘=’?
    - Assignment is different from definition and comparison
    - I = I + 1 ; I := I + 1

Imperative Constructs

- Control-flow
  - – All imperative constructs alter flow of control (except assignment)
  - – Has *if-then-else*
  - – *for*-loop replaces *do*-loop
- No input/output constructs
  - – I/O was left to be handled by platform-dependent library calls

Name Structures

- Algol-60 introduces the compound statement
  - – Where 1 statement is allowed, more can be used, using begin-end

```
for i := 1 step 1 until N do
      sum := sum + Data[i]

for i := 1 step 1 until N do
  begin
      sum := sum + Data[i];
      Print Real (sum)
  end
```

  - – Also, the body of a procedure is a single statement

Syntax - Program

- <program> ::= <block> | <compound statement>

- <block> ::= <unlabelled block> | <label>: <block>
- <compound statement> ::= <unlabelled compound> | <label>: <compound statement>

- <unlabelled compound> ::=
  begin <compound tail>
- <unlabelled block> ::=
  <block head> ; <compound tail>

Syntax - Block

- <block> ::= <unlabelled block> |
  <label>: <block>
- <unlabelled block> ::=
  <block head> ; <compound tail>
- <block head> ::= begin <declaration> |
  <block head> ; <declaration>
- <compound tail> ::= <statement> end |
  <statement> ; <compound tail>

Syntax - Statement

- <compound statement> ::= <unlabelled compound> |
  <label>: <compound statement>
- <unlabelled compound> ::= begin <compound tail>
- <compound tail> ::= <statement> end | <statement> ; <compound tail>
- <statement> ::= <unconditional statement> | <conditional statement> |
  <for statement>
- <unconditional statement> ::= <basic statement> |
  <compound statement> | <block>
- <basic statement> ::= <unlabelled basic statement> |
  <label>: <basic statement>
- <unlabelled basic statement> ::= <assignment statement> |
  <go to statement> | <dummy statement> |
  <procedure statement>

Name Binding

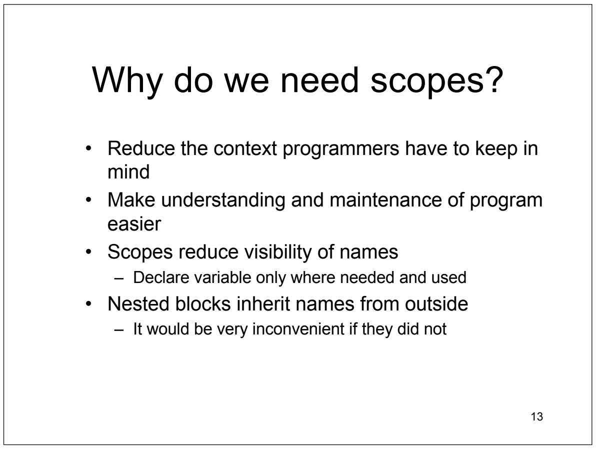
- Fortran binds a variable to a single memory location statically
- Algol-60 included the results of research done on name structures, which were problematic in Fortran
  - Sharing of data between subprograms
  - Parameter passing modes
  - Return values
  - Dynamic arrays
- Result of research: block structure

Blocks Define Nested Scopes

- Fortran
  - Had local and global declarations only
  - Had to redeclare using COMMON to share
- Algol-60
  - Introduces blocks

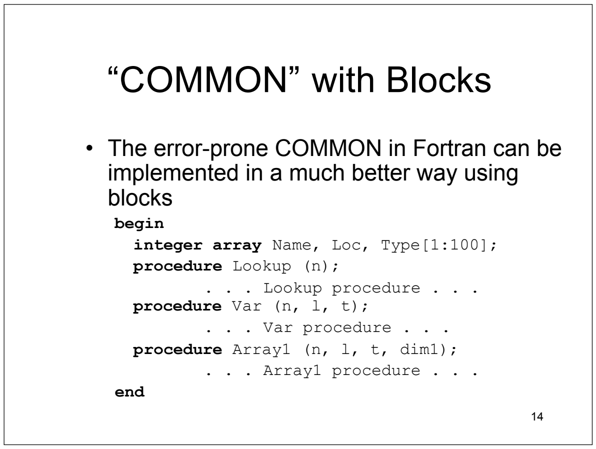
```
begin
  declarations;
  statements
end
```

  - Compound statements do not have ‘`declarations`’.
  - All declarations are visible to all statements in the block
  - Since statements can be blocks, scopes can be nested



Why do we need scopes?

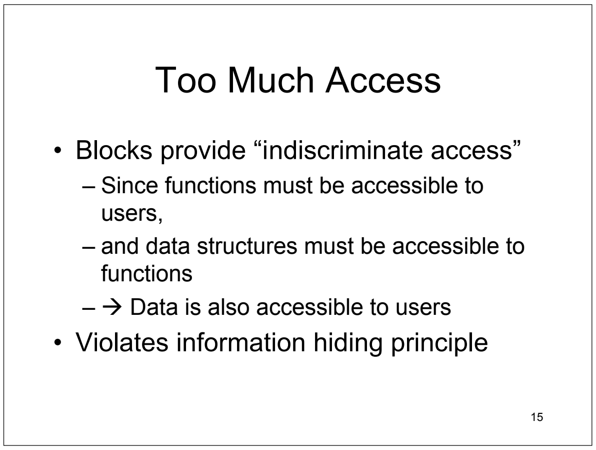
- Reduce the context programmers have to keep in mind
- Make understanding and maintenance of program easier
- Scopes reduce visibility of names
  - Declare variable only where needed and used
- Nested blocks inherit names from outside
  - It would be very inconvenient if they did not



“COMMON” with Blocks

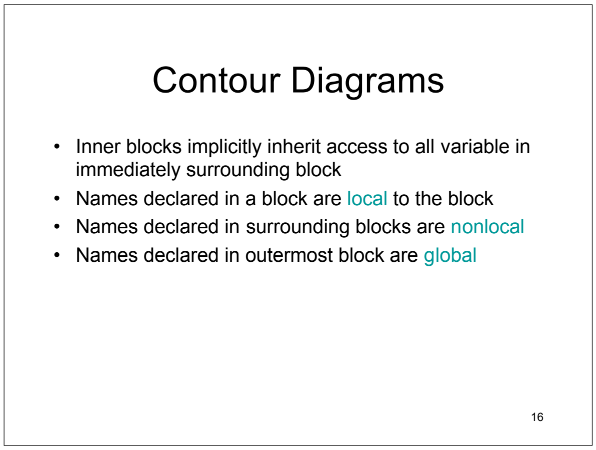
- The error-prone COMMON in Fortran can be implemented in a much better way using blocks

```
begin
  integer array Name, Loc, Type[1:100];
  procedure Lookup (n);
         . . . Lookup procedure . . .
  procedure Var (n, l, t);
         . . . Var procedure . . .
  procedure Array1 (n, l, t, dim1);
         . . . Array1 procedure . . .
end
```



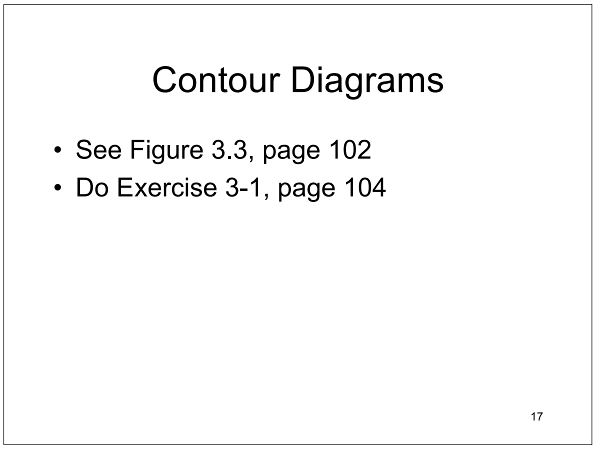
Too Much Access

- Blocks provide “indiscriminate access”
  - Since functions must be accessible to users,
  - and data structures must be accessible to functions
  - → Data is also accessible to users
- Violates information hiding principle



Contour Diagrams

- Inner blocks implicitly inherit access to all variable in immediately surrounding block
- Names declared in a block are local to the block
- Names declared in surrounding blocks are nonlocal
- Names declared in outermost block are global



Contour Diagrams

- See Figure 3.3, page 102
- Do Exercise 3-1, page 104