CS 4100
Pascal Highlights

April 1, 2011

Based on slides by Istvan Jonyer
Book by MacLennan

Procedure Constructor

- Procedure declaration in Pascal has a strict structure

```
procedure <name>(<formals>)
   <label declarations>
   <const declarations>
   <type declarations>
   <var declarations>
   <procedure and function declarations>
begin
   <statements>
end
```

- Similar to Algol's
  - Scope essentially the same
    - Declarations: entire block including declarations and statements
    - Formals: local declarations and statements
- Names bound before they are used to support one-pass compilation

Mutual Recursion

```
procedure P(…);
begin
   .
    Q(…);
   .
end;
procedure Q(…);
begin
   .
    P(…);
   .
end;
```


Procedure Constructor

- Opposite of top-down
  - Uppermost procedures first, then lower ones they call
- Mutual recursion
  - Cannot define both procedures before one is called
- Pascal's solution
  - "`forward`" declaration of procedures allows recursion, and observation of structure principle

```
procedure Q(…); forward;
```

No Blocks

- Pascal eliminates Algol's blocks
  - Compound statements but no blocks
  - Variable declarations are only allowed before *begin* in procedures and functions
  - Simplifies name structures
  - Complicates efficient use of memory
    - Storage shared only between disjoint procedures

5

Control Structures

- Pascal includes more control structures than Algol-60, but they are simpler
  - Provides simple I/O
  - Introduces more structured control structures (structure principle)
    - 1-entry point 1-exit point controls
  - Includes goto (rarely needed)
  - Includes recursive procedures

6

`for`-Loop is Austere

- Pascal removes the baroque for loop, in favor of one simpler than Fortran's

```
for <name> := <exp> {to|downto} <exp> do
    <statement>
```

  - Only step size of 1 is allowed (+1 & -1)
    - May be too restrictive
  - Bounds are computed once, on entry
    - Called *definite iterator*
      - Always executes a definite number of times unless goto

7

Leading & Trailing Decision Loops

- Indefinite iterators:
  - Loop is controlled by condition, not counter
  - Condition is tested each time
    - Versus pre-computed in *for*-loop
- Leading Decision loop

```
while <condition> do <statement>
```

- Trailing Decision loop

```
repeat <statement>+ until <condition>
```

- Mid-Decision loop
  - Can be implemented using "**`while true do`**" and **`goto`**

8

Pascal's `case`-Statement

- Pascal introduces the labeled, structured case-statement

```
case <expression> of
  1:    begin <statements> end;
  2, 3: begin <statements> end;
  4:    begin <statements> end;
  ...
end case;
```

  – This case-statement is *self-documenting*

Labels in `case`-Statement

- Case labels can be labels from enumeration types

```
case nextFlight.status of
  inAir:     begin <statements> end;
  onGround:  begin <statements> end;
  atTerminal:begin <statements> end;
end case;
```


Parameter Passing

- Pass by value
  – Exactly like before, in Algol-60
- Pass by reference
  – Allows output parameters
  – Replaces pass by name
  – Only allows meaningful variables to be written into (unlike Fortran)

Pass as Constant

- Pass as constant was originally specified instead of pass by value
  – Like pass by value, but parameter could not be modified in callee
    - Safe
  – Implemented as pass by reference
    - Efficient
  – Replace by pass by value, since pass as constant can be circumvented using scoping (p 202)
    - C++ provides this functionality by explicit pass by reference and *const* definitions (`f(const int &a)`)

Two Orthogonal Issues

- Input vs output parameters
- Copy value vs pass address
- Decisions should be separated

13

Goals

- Main goal: good teaching language
  - Reliability
  - Simplicity
  - Efficiency
- Successful!
- Third Generation

14