CS 4100
Pascal Highlights

March 26, 2012

Based on slides by Istvan Jonyer
Book by MacLennan

1

Array Types

- Arrays are more general than Algol's
  - Base type of arrays can be non-primitives
  - Index types are introduced
  - Subscripts can be other than integers
    - Char, subrange, enumerated types

```
var A: array [1..100] of real;
var Occur: array [char] of integer;
var HoursWorked: array [Mon..Fri] of 0..24;

for day := Mon to Fri do
  TotalHours := TotalHours + HoursWorked[day];
```

2

Dimensions

- Only single-dimension arrays are allowed!!!
- However:
  - Base type of array can be another array!!!

```
var M: array [1..20] of array [1..100] of real;
```

  - Dereferencing: `M[3][5]`
- *Syntactic sugar*:

```
var M: array [1..20, 1..100] of real;
M[3, 5]
```

(Doesn't affect functionality, sweeter for human use.)

3

Static Arrays Only

- Algol's dynamic arrays are not supported
  - Type checking is done at compile time
  - Array bounds are part of array type
  - Hence, only static arrays are supported

4

Record Types

- Pascal provides the ability to group heterogeneous data
  - Versus homogeneous, using arrays
  - Can contain any other type, even other records

```
type person =
    record
          name: string;
          age: 16..100;
          salary: 10000..100000;
          sex: (male, female);
          hireDate: date;
    end;
    string = array [1..30] of char;
```

5

Dereferencing Records

- Dereferencing is done using the '.'

```
var today: date;
newhire.age := 25;
newhire.hireDate := today;
newhire.hireDate.month := Mar;
if newhire.name[1] = 'A' then …
employee[en].hireDate.year := 2004;
```

- Opening one record for multiple access

```
with newhire do
  begin
    age := 25;
    hireDate := today;
    hireDate.month := Mar;
  end;
```

6

Variant Records

- Pascal supports saving storage using variant records; allows alternative structures
  - Not all components of a record may be used at the same time
    - E.g.: Plane altitude and location on ground
  - C: union
    - Union is unsafe as it allows access to any member
  - Pascal attempts to solve this security problem
    - Access only members allowed by *tag field*
    - Initialization not required after tag value change, so type system can be circumvented after all…

7

Variant Record Example

```
type plane = record
  flight: 0..999;
  equipment: (B727, A343, B747);
  case status: (inAir, taxi, atTerminal) of
  inAir: (
      altitude:   0..999999;
      heading:          0..359);
  taxi: (
      location:   airport;
      runway:     runwayNumber);
  atTerminal: (
      parked:     airport;
      gate:       1..100);
end;
```

8

Pointers

- Pascal provides *typed pointers*, which are more secure than untyped ones

```
var p: ↑real;
    x: real;
    c: char;
begin
    new(p);
    p↑ := 3.14159;
    c := p↑;          {Illegal!}
end;
```

  - If P was untyped (`p: ↑pointer`), assignment to c would be allowed (and meaningless)

9

Type Equivalence

- Type checking requires that only variables with identical types can be compared/ assigned to each other
- What does 'identical' mean?
  - Structural equivalence
    - Types having the *same structure* are identical

```
var x: record id: integer; w: real end;
var y: record id: integer; w: real end;
```

  - Name equivalence
    - Types having the *same name* are identical

10

Structural equivalence

```
type person = record id:integer; weight real; end
type car = record id:integer; weight real; end
var x: person;
var y: car;
x:= y;
```

  - Legal by structural equivalence
  - Probably don't want
  - Name equivalence fixes this - person and car are different names

11

Name Structures

- Name binding mechanisms in Pascal
  - Constant bindings
  - Type bindings
  - Variable bindings
  - Procedure and function bindings
  - Implicit enumeration bindings
  - Label bindings

12

Constants

- Pascal introduces constant declarations

```
const <name>=<constant>;
const MaxArray = 100;
```

  - Allows the naming of constants in program
  - Numbers should not be used in programs
- Application of Abstraction Principle

13

Constants - Limitations

- Constant cannot be described by an expression
  - Illegal:

```
const MaxArray = MaxData - 1;
```

- Expressions are not allowed in variable and type declarations
  - Illegal:

```
var A: array [0.. MaxData – 1] of real;
```

14

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

15

Mutual Recursion

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

16

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;
```

17

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

18

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

19

`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

20

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`**

21

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*

22

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;
```

23

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)

24

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)`)

25

Two Orthogonal Issues

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

26

Goals

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

27