V22.0490.001 Special Topics: Programming Languages

B. Mishra New York University.

Lecture # 5

—Slide 1— PASCAL

Language Survey 1

- Invented by Nicklaus Wirth between 1969 and 1970
- Motivation:
 - —Language for teaching programming
 - —Reliable and Efficient Language
- Characterized by simplicity
 - —"Streamlined" Algol
 - (Simpler Data Structure)
 - —Added User-defined data types
- Algol-like (Declarations and Imperative)

—Slide 2—

Syntax (Declarations)

• Variables:

var x, y, z: integer;

• Procedure:

procedure foo(x, y: char; var z: real);
 var
begin
 ...
end;

• Function:

function bar(a, b: integer): integer; var x, y: real; begin ... end;

—Slide 3—

Imperative Statements

• Assignment:

a := b + c

• Control Structure:

```
for i := 1 to N do ...
while (i = 0) do ...
repeat ... until (i = 0)
case i of
   1: ...;
   2: ...;
end;
```

• Like Algol, the body of each control structure can be only one (simple or compound) statement.

—Slide 4— Compound Statements

• Syntax:

```
begin
    x := y;
    y := y * 3;
end;
```

• Any where a single statement can go, a compound statement can go (like Algol)

—Slide 5—

Block Structure

• Pascal is Block Structured —(Procedures are nested)

• ...And statically scoped

—(Procedures are evaluated in the environment of their definition).

- Compound statements in Pascal do not define blocks—Blocks are only defined by procedure declaration
- Following is not valid

```
begin
...
var x, y: integer;
...
end;
```

—Slide 6—

Example:Pascal

procedure foo(var x, y: integer);
 var a, b, c, d: integer;
begin
 ...
end;

—Slide 7—

Compound Statement: Contd

- A compound statement serves only one purpose in Pascal
 - Greater orthogonality
- But
 - 1. Not as space efficient
 - —All local variables stay in activation record
 - —Because they are allocated for the life of the procedure while they may be only needed for the life of the block
 - 2. Makes program modification difficult —Cannot insert blocks

—Slide 8— *Example*

—Slide 9—

Parameter Passing

• Pascal gives you a choice of

- Call by value (val)

- Call by reference (var)

procedure foo(x, y: integer; var z: real);

--x, y are *val* parameters--z is *var* parameter

—Slide 10—

Parameter Passing: Examples

• Call-by-Value:

```
procedure foo(x, y: integer);
begin
    x := 1;
    y := x * 3;
end;
```

foo has no effect on the program that calls it

• Call-by-Reference:

```
function foo(var x: integer): integer;
begin
    x := 6;
    foo := x;
end;
```

foo is not a pure function; \mathbf{x} changes as a result of the side effect

—Slide 11—

$Side \ Effects$

- A side effect is an assignment in a procedure or function call that is not *obvious* to the caller.
- A function call is only expected to affect the arguments.
- A pure function (e.g., sin(x) or sqrt(x)) is expected to return a value and not change any other variable
- With proper care, side effect can be useful

—Slide 12—

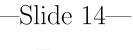
Example

```
program StackManip;
  integer stack[20], index;
  procedure push(x);
    integer x;
    begin
      stack[index] := x;
      index := index + 1;
    end
  function pop(): integer;
    begin
      index := index - 1;
      pop := stack[index];
    end;
  begin
    y := pop();
    push(3);
  end.
```

—Slide 13— User Defined Types

- Pascal's big advance:
 - Hierarchical Type Structure
 - Type composed of other types
- Defining types in Pascal

type <type-name> = <type-definition>



Types

- Type has two components

 A set, S of elements
 A set of operations on S
- Language-defined types: integer: {-2³¹..2³¹}, {+, -, *, /, ..., }; boolean: {T, F}, { not, and, or, ..., } real: {r|r is real }, {+, -, *, /, sqrt, ..., };
- Larger domain constructs: Array, Product, Union,...

—Slide 15—

$Type \ Abstraction$

• A problem may require representations, that cannot be properly abstracted by integers, reals etc.

—Simulating the behavior of a car

- Pascal provides type abstractions to create abstract data types
 - Information Hiding
 - Machine independent
- Pascal provides primitive data types and orthogonal mechanisms for composing new composite types from the primitive types.

—Slide 16— Primitive Types

- The primitive types supplied by Pascal: — Real, Integer, Character, Boolean
- User-defined primitive type: *Enumerated* type

—Slide 17—

Primitive Types: Enumerated Types

• Enumerated Types:

```
type shortweek = (Monday, Tuesday, Wednesday);
```

- Describe whole set S of elements
- Operations

 Enumerated types have ordered set of elements:

=, <, <=, >, >=, <>, :=, succ, pred, ord

Other Operations:
 —Other user-defined operations on the type are also allowed

—Slide 18—

Primitive Types: Subrange Types

- Allows one to specify a subset $S' \subset S$ of another set S, without explicitly listing all the elements of S.
- Example

type DayOfMonth = 1 .. 31;

—The subset is specified by giving the min and the max elements.

type week = (Mon, Tue, Wed, Thu, Fri, Sat, Sun); type weekdays = Mon..Fri;

• Operations on the derived type: Same as the base types

var x : DayOfMonth;

One can apply the same operations as the ones defined for the integers: $\{+, -, *, /, \ldots, \}$;

—Slide 19— Subrange Types

• Which Type is **DoWop** derived from?

type foo: (bar, baz, bop, bif)
 boo: (baz, baf, bir, bop)
 DoWop: baz..bop;

—Ambiguous

- In Pascal, the *enumerated types* must be disjoint. Thus, **foo** and **boo** are illegal.
- Thus there is no ambiguity.
- Ada 95 resolves this problem in a completely different manner.

[End of Lecture #5]