V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 6

—Slide 1—

PASCAL
Aggregate Types

- Each element of an aggregate type is composed of another type
- Sets

```
type foo: set of Mon .. Thu;
     bar: set of shortweek;
```

—`shortweek` must be a finite discrete primitive type

- The base type can be any enumerated type, subrange type, Boolean, Character or Integer
— But not Real

—Slide 2—

Aggregate Types (Contd)

- In Pascal, sets are implemented by bit vectors.
- Each element of the base type is represented by a bit.
- You cannot have infinite bit vectors.
  — The base type must be finite. — The base type must be primitive
  — The base type cannot be Real or another aggregate type.

—Slide 3—

Using Set Types: Example

```
var y, x, z: set of shortweek;
begin
  z := [Tue, Thu];
  x := [Mon, Wed];
  y := x * z;     (*Set Intersection*)
  x := y + z;     (*Set Union*)
  ...
end;
```

—Operations also include:
Set Difference `-`; Set Equality and Inequality `=`, `<>`; Subset and Superset `<=`, `>=`

—Slide 4—

Array Types

- Syntax

```
type <name> = array [ <finite-type> ]
                   of <type>
```

—Index Type: enumerated, subrange or finite primitive type
—Base Type: any type

- An array type is a *homogeneous aggregate type*, since all entries are of the same type

- Pascal arrays are *static*
— Size is fixed at the compile time

—Slide 5—

Multidimensional Arrays

- Multidimensional Arrays are Arrays of Arrays
- Example: Following are equivalent

```
var a: array [1..10]
          of array [1..10] of integers;

var a: array [1..10, 1..10] of integers;
```

- This is only for syntactic convenience.

—Slide 6—

Examples

- Examples

```
type foo = array [1..10] of integers;  (*simple*)
     string = array [1..10] of char;
     MonthLength = array [Jan..Dec] of DayOfMonth;
```

- In the last type definition, the indexes (a subrange) and the base type (an enumerated type) must have already defined.

```
var x: foo;
    y, z: string;
    w: array [Mon..Fri] of Mon..Fri;
```

—Slide 7—

Record Types

- Syntax

```
type <name> =
  record
    speed     : integer;
    direction : (N, S, E, W);
    color     : red .. violet;
  end;
```

- A record type is a heterogeneous aggregate type
  — More general than homogeneous aggregate type
  — Can be composed of elements of different types

—Slide 8—

Records (Contd)

- Arrays

```
A [ <exp> ]      (* Less general *)
```

— Allows arbitrary expression `<exp>` evaluating to a value of the index type

- Records

```
John.Age         (* More general *)
John.Address
```

— Fields must be named and fixed at compile time

—Slide 9—

Variant Records

- The available fields are determined by a (tag) value computed at run time
- Example

```
type car =
  record
    status    : (driving, parked);
    road      : (I95, NYSThruway, NJTurnpike);
    ParkSpace : 1 .. 100;
  end;
```

— But a car cannot be both **`driving`** and **`parked`** at the same time!
— One of the fields contains the correct information
— This also presents a "security" loophole

—Slide 10—

Variant Records (Contd)

- Variant Record (Union type):

```
type car =
  record
  case status    : (driving, parked) of
    driving: (road: (I95, NYSThruway, NJTurnpike));
    parked:  (ParkSpace : 1 .. 100);
  end;
```

— If a `car` is `driving` its field is `road`
— If a `car` is `parked` its field is `ParkSpace`
— It creates a typing loophole in Pascal.

—Slide 11—

Pointer Types

- The value of a pointer is the address of a variable (L-value)

```
var p : ^real;
    q : ^person;
```

- Example
  — Declare an element of a linked list.

```
type foo =
  record
    name  : string;
    next  : ^foo;
  end;
```

— Singly-linked list

```
     +---+----+   +---+----+   ++
foo -->[   |next]-->[   |next]-->||
     +---+----+   +---+----+   ++
```

—Slide 12—

Typing

- Typing is the set of rules determining the correct use of types
  — Type checking is the process of deciding if types are used correctly
- What constitutes a valid use of types? — Using only the operations that apply to the elements of the type of the operands.
- Example

```
var x: integer;
    y: char;
begin
    x := 5 + 6;
    y := x + 1;   (* type mismatch *)
```

—Slide 13—

Strongly Typed Languages

- A language that allows one to completely determine if types are being used correctly
  — Can determine the type of every object in the program
  — Example: Algol, Pascal,...
- *Weakly Typed Language*: A language that is not strongly typed
  — Example: C, ... — Compiler may not be able to perform type-checking
  — Type-checking must be deferred to run-time.
- Interpreted languages must be weakly-typed!

—Slide 14—

Type Equality

- When are two objects of the same type?
- Consider for instance,

```
car x: record
         MyId: integer;
       end;
    y: record
         MyId: integer;
       end;
```

— Are `x` and `y` of the same type?

- Two approaches:
  — *Name Equivalence*: Have the same type as their type names. $\mathtt{x} \not\equiv \mathtt{y}$
  — *Structural Equivalence*: Have the same type as their type structures. $\mathtt{x} \equiv \mathtt{y}$

—Slide 15—

Name Equivalence

- Pascal uses *name equivalence* (+ *declaration equivalence*)
- Two types can be explicitly declared to be equivalent.
- Problem with name equivalence: Consider

```
var x: 1..30;
    y: integer;
begin
    ...
    y := 25; x := y;
    ...
    y := 100; x := y;
    ...
end;
```

— Note, `x` and `y` are not of the same type?

—Slide 16—

Type Violation Through Variant Records

- Problems with variant records

```
var x: record
       case a : (1, 2) of
         1: (r: real);
         2: (c: char);
       end;
    y: 1..2;
begin
  ...
  y := y+1;
  x.a := y;
  x.r := 1.15; (*improper usage, if y=2*)
  ...
end;
```

— Compiler cannot detect improper use!

- Is Pascal *strongly-typed*?

—Last Slide—

Summary

Pascal Design

- Good!
  1. Simple, useful for pedagogic purposes
  2. Block-structured (Algol-like)
  3. User defined types
- Bad!
  1. Too simple for large applications
  2. No modules or facilities for separate compilation

[End of Lecture #6]