Record Types

• A record is a possibly **heterogeneous aggregate** of data elements in which the individual elements are identified by names

• Design issues:
  o What is the syntactic form of references to the field?
  o Are elliptical references allowed
Definition of Records in COBOL

- COBOL uses level numbers to show nested records; others use recursive definition

  01 EMP-REC.
    02 EMP-NAME.
      05 FIRST PIC X(20).
      05 MID PIC X(10).
      05 LAST PIC X(20).
    02 HOURLY-RATE PIC 99V99.
Definition of Records in Ada

• Record structures are indicated in an orthogonal way

```ada
type Emp_Rec_Type is record
    First: String (1..20);
    Mid: String (1..10);
    Last: String (1..20);
    Hourly_Rate: Float;
end record;
Emp_Rec: Emp_Rec_Type;
```
References to Records

• Record field references
  1. COBOL
      field_name OF record_name_1 OF ... OF record_name_n
  2. Others (dot notation)
      record_name_1.record_name_2. ... record_name_n.field_name

• Fully qualified references must include all record names

• Elliptical references allow leaving out record names as long as the reference is unambiguous, for example in COBOL

FIRST, FIRST OF EMP-NAME, and FIRST of EMP-REC are elliptical references to the employee’s first name
Operations on Records

• **Assignment** is a common operation
  o if the types are identical

• Ada allows **record comparison**

• **COBOL** provides `MOVE CORRESPONDING`
  o Copies a field of the source record to the corresponding field in the target record
Evaluation and Comparison to Arrays

- Access to array elements is much slower than access to record fields, because subscripts are dynamic (field names are static)
- Dynamic subscripts could be used with record field access, but it would disallow type checking and it would be much slower
Offset address relative to the beginning of the records is associated with each field.
Union Types

• A union is a type whose variables are allowed to store different type values at different times during execution

• Design issues
  - Should type checking be required?
  - Should unions be embedded in records?
Discriminated vs. Free Unions

- Fortran, C, and C++ provide union constructs in which there is no language support for type checking;
  - Called *free union*
- Type checking of unions require that each union include a type indicator called a *discriminant*
  - Supported by Ada
Ada Union Types

type Shape is (Circle, Triangle, Rectangle);

type Colors is (Red, Green, Blue);

type Figure (Form: Shape) is record
    Filled: Boolean;
    Color: Colors;
    case Form is
      when Circle => Diameter: Float;
      when Triangle =>
        Leftside, Rightside: Integer;
        Angle: Float;
      when Rectangle => Side1, Side2: Integer;
    end case;
end record;
Ada Union Type Illustrated

A discriminated union of three shape variables
Evaluation of Unions

• Free unions are unsafe
  o Do not allow type checking

• Java and C# do not support unions
  o Reflective of growing concerns for safety in programming language
Name Type Equivalence

- *Name type equivalence* means the two variables have equivalent types if they are in either the *same declaration* or in declarations that use the same type name.
Structure Type Equivalence

- Structure type equivalence means that two variables have equivalent types if their types have identical structures.
- More flexible, but harder to implement.
Type Equivalence (continued)

- Consider the problem of two structured types:
  - Are two record types equivalent if they are structurally the same but use different field names?
  - Are two array types equivalent if they are the same except that the subscripts are different? (e.g. [1..10] and [0..9])
Type Equivalence (continued)

- Are two enumeration types equivalent if their components are spelled differently?

- With structural type equivalence, you cannot differentiate between types of the same structure (e.g. different units of speed, both float)