§9.7–9.10: Records

•*devo*

4 Nov 2005
CMPT14x
Dr. Sean Ho
Trinity Western University

Reminders:

- ***journals in folder***
- ***Quiz today***

Quiz ch8 (7 questions, 20 marks, 10 minutes)

- Convert 1101 1011 from binary to hexadecimal.
- If 101C = 'A', what is 110C?
- Express 110C using the CHR() notation.
- Express 2Mb/sec in bytes/sec.
  - (you may express your answer in powers of 2)
- In your own words, describe the difference between CAST and VAL.
- What M2 type do data storage units have, and in what library is this type found?
- What M2 library is used to open/close rewindable sequential text streams?

Quiz ch8 answers

- 1101 1011: convert one nibble at a time
  - = 0DBH
- 'A' = 101C = CHR(65): first letter
  - 110C = CHR(72) = eighth letter = H
- 2Mb/s
  - = $2^1 2^{20}$ bit/s = 2^{21} bit/s = 2^{18} byte/s
- CAST: does not modify bit pattern, unsafe
  - VAL: converts value, safe type conversion
- data storage units: SYSTEM.LOC
- rewindable sequential text streams: SeqFile

Modula-2 Types

- Atomic types
  - Scalar types
    - Real types (REAL, LONGREAL)
    - Ordinal types
      - **Whole number types (INTEGER, CARDINAL)**
      - **Enumerations (5.2.1)**
      - **Subranges (5.2.2)**
- Structured (aggregate) types
  - Arrays (5.3)
    - Strings (5.3.1)
  - Sets (9.2–9.6)
  - Records (9.7–9.12)

today

Review of last time (9.1–9.6)

- Using sets
  - Defining a set type
  - Declaring a set variable
  - Constructing a set
- Operations with sets
  - Set operations: IN, +, *, –, /
  - INCL/EXCL
  - Set comparisons: =, <>, >=, <=
- Bitsets and packed sets

What's on for today (9.7–9.10)

- Records
  - Defining record types
  - Fields
  - Initializing record variables
  - WITH
- Using records and arrays
  - Example: Class of students
- Output of aggregate data

Records

- All members of a set have to be the same type
- An M2 record abstracts an aggregate of related data (fields) of various types

```
TYPE
    EmployeeRecord =
        RECORD
            name : ARRAY [0 .. 255] OF CHAR;
            age : CARDINAL;
            salary : REAL;
        END;
VAR
    emp1 : EmployeeRecord;
emp1.name := “Joe Smith”;
```

Record fields

- A field in a record can have any type, including another record type:

```
EmployeeRecord =
    RECORD
        name : ARRAY [0 .. 255] OF CHAR;
        age : CARDINAL;
        salary : REAL;
        birthdate =
            RECORD                      (* anonymous type *)
                year : CARDINAL;
                month : [1 .. 12];
                day : [1 .. 31];
            END;
    END;
emp1.birthdate.month := 6;
```

Using records

- We can initialize records by filling in each of its fields:

```
emp1.name := “Joe Smith”;
emp1.birthdate.month := 6;
```

  - Uninitialized fields are like uninitialized vars

- We can assign a whole record to another:

```
emp2 := emp1;
```

- But we cannot compare whole records:

```
IF emp1 = emp2 ...          (* error! *)
```

Records and WITH (scope) blocks

- As a shorthand for

```
emp1.name := “Joe Smith”;
emp1.birthdate.month := 6;
```

- We can also write

```
WITH emp1
    DO
        name := “Joe Smith”;
        birthdate.month := 6;
        WITH birthdate
            DO
                year := 1985;
            END;
    END;
```

Records vs. arrays (or both?)

- Say we're keeping track of a class of students:
  - For each student, store name, student ID, and marks for each of four exams
- We could implement this with separate arrays:
  - One array for all the names
  - Another array for all the student IDs
  - One multidimensional array for all exam marks
- Or we could use an array of records:
  - Each record stores everything for one student
  - 3 fields: name, ID, exam marks

Array of student records

```
TYPE
    NameString = ARRAY [0 .. 255] OF CHAR;    (* string *)
    Student =
        RECORD
            name : NameString;
            ID : CARDINAL;
            marks : ARRAY [1 .. 4] OF REAL;
        END;
    Class = ARRAY [1 .. 30] OF Student;
VAR
    cmpt145 : Class;
BEGIN
    WITH cmpt145[1]            (* one student at a time *)
        DO
            marks[1] := 95.1;       …
```

Storing aggregate data to file

- We know how to output atomic data to files in text form:

```
WholeIO.WriteCard (cid, class[1].ID, 0);
```

- To output aggregate data to files,
  - We could devise our own text format:

```
TextIO.WriteString (cid, “Student ID:”);
WholeIO.WriteCard (cid, class[1].ID, 0);
```

  - But easier and more space-efficient to output as binary:

```
RawIO.Write (cid, class[1]);    (* output 1st student *)
RawIO.Write (cid, class);       (* output whole class *)
```

Review of today (9.7–9.10)

- Records
  - Defining record types
  - Fields
  - Initializing record variables
  - WITH
- Using records and arrays
  - Example: Class of students
- Output of aggregate data

TODO items

- Lab 7 due next week: 8.13 #(53 / 60 / 62)
- HW due next Wed: 9.14 #30
- Quiz ch9 next Wed
- Reading: through §9.10 for Fri