§9.1-9.6: Sets

•*devo*

3 Nov 2005
CMPT14x
Dr. Sean Ho
Trinity Western University

Reminders:

- ***journals in folder***

Review of last time (8.6-8.12)

- Sequential streams: StreamFile driver
  - StringIO, WholeIO, RealIO libraries
- Rewindable streams: SeqFile driver
  - Reread and Rewrite
  - File modes: read/write/old
- Binary streams: RawIO driver
- Standard Channels (StdInChan, StdOutChan)
- Low-level device-independent I/O: IOChan
    - (just be aware that StreamFile/SeqFile/etc. use IOChan for even lower-level stuff)

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) ← *today*
  - Records (9.7-9.12)

What's on for today (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

Sets

- An M2 set is a collection of items of the same scalar non-real type, without regard to order:

```
TYPE
    UpperCaseSet = SET OF ["A" .. "Z"];
    DigitSet = SET OF [0 .. 9];
VAR
    vowels : UpperCaseSet;
    octalDigits : DigitSet;
BEGIN
    vowels := UpperCaseSet {"A", "E", "I", "O", "U"};
    octalDigits := DigitSet {0 .. 7};
```

- Note distinction between type and variable
- Constructors initialize the set variables

Set operations

```
vowels := UpperCaseSet {"A", "E", "I", "O", "U"};
```

- Test set membership:

```
IF char IN vowels …
```

- Set union: (OR)

```
nameLetters := UpperCaseSet {"S", "E", "A", "N"};
union := vowels + nameLetters;
    (* union = {"A", "E", "I", "O", "U", "S", "N"} *)
```

- Set intersection: (AND)

```
intersect := vowels * nameLetters;
    (* intersect := {"A", "E"} *)
```

Set operations, cont.

```
vowels := UpperCaseSet {"A", "E", "I", "O", "U"};
nameLetters := UpperCaseSet {"S", "E", "A", "N"};
```

- Set difference: (AND NOT)

```
diff := vowels - nameLetters;
    (* diff = {"I", "O", "U"} *)
```

- Set symmetric difference: (XOR)

```
symdiff := vowels / nameLetters;
    (* symdiff := {"I", "O", "U", "S", "N"} *)
```

INCL/EXCL: analogues to INC/DEC

vowels := UpperCaseSet {“A”, “E”, “I”, “O”, “U”};

- INCL (vowels, “Y”)
  - Same as: vowels := vowels + UpperCaseSet {“Y”};
- EXCL (vowels, “E”)
  - Same as: vowels := vowels – UpperCaseSet {“E”};

Set comparisons

- Only sets of same type can be compared
  - Otherwise compile-time error, type mismatch

    vowels := UpperCaseSet {“A”, “E”, “I”, “O”, “U”};

    nameLetters := UpperCaseSet {“S”, “E”, “A”, “N”};
- Equal, not-equal, superset, subset:
  - IF vowels = nameLetters ...
  - IF vowels <> nameLetters ...
  - IF vowels >= nameLetters ...
  - IF vowels <= nameLetters ...
- No proper subset operator (> or <)

Bitsets

- A bitset is a way of thinking of a binary number:
  - 01101100 has 1s in positions 6, 5, 3, 2
  - Rightmost position is position 0
  - Think of this binary number as set {6, 5, 3, 2}

```
VAR myBitset : BITSET;
myBitset := CAST (BITSET, 108);
IF 5 IN myBitset ...
```

- Can think of BITSET as an implicit type:

```
TYPE BITSET = SET OF [0 .. BitsPerBitset-1];
BitsPerBitset := SIZE (BITSET) * SYSTEM.BITSPERLOC;
```

Packed Sets

- All BITSETs have fixed size
  - Usually same as system word size (e.g. 32 bits)
- Packed sets let you set the size:

```
TYPE BigBitset = PACKEDSET OF [0 .. 99];
```

- Packed sets and bitsets can be shifted/rotated:

```
10011 ----- (shift 1)      -----> 00110
10011 ----- (shift -1)     -----> 01001
10011 ----- (rotate 1)     -----> 00111
10011 ----- (rotate -1)    -----> 11001
```

- Packed sets are useful for examining the binary storage of complex data types

(Preview of) 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";
```

Review of today (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

TODO items

- Quiz ch8 tomorrow
- Lab 7 due next week: 8.13 #(53 / 60 / 62)
- Reading: through §9.10 for Fri