Structured data types

• So far:
  – **scalar** (single value) data types
  – structured data type: **array**

• **records**: data structure that collects together into one unit several related items of data
  – Name, phone number, sex, age, and weight
  – Day number, month name, and year number
  – ...

Reading: FK pp. 367-384, 415-420, 214-217

Lecture 16
Oct 8 2003
Arrays

- Access elements using Indices
  - Single Dimension arrays A(I)
  - Two dimensional arrays A(I,J)
  - N dimensional array A(i_1, i_2,..,i_n)

- Loops can be used to access control to elements.

```plaintext
for I in 1 .. N loop
    Get (A(I));
end loop;
```

```plaintext
for I in 1 .. M loop
    for J in 1 .. N loop
        Put (B(I,J));
    end loop;
end loop;
```

Records

- To use records we need to know:

  1. How to design a record
  2. How to declare record types and variables
  3. How to use a record
1. Designing Records

- **To design a record:**
  
  - **identify** the items of data that are relevant in this application
  
  - use a **data structure diagram** to show the relevant information
    
    - decide on **names** for the overall structure, and for the individual fields
  
  - determine the **data types** of the fields

---

**Example1 Fitness club**

<table>
<thead>
<tr>
<th>persons</th>
<th>name</th>
<th>phone</th>
<th>sex</th>
<th>age</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>names;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>phones;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sex</td>
<td>sexes;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>age</td>
<td>ages;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>weights;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

name : names; -- string sub-type
phone : phones; -- string sub-type
sex : sexes; -- enumerated type
age : ages; -- integer sub-range
weight : weights; -- float sub-type
2. Declaring records

- Form of declaration:

```plaintext
-- declaration of record data type
type record_type_name is record
  field_name_1 : field_type_1;
  field_name_2 : field_type_2;
  -- various fields in the record
end record;
```

- Example - positional aggregate:

```plaintext
average_male : constant persons :=
  ("Mr. A Average",
   "",
   male, 25, 72.5);
```

- Example - named aggregate:

```plaintext
average_female : constant persons :=
  (name => "Ms. A Average",
   phone => "",
   sex => female,
   age => 21,
   weight => 62.0);
```
3. Using records

- To refer to an entire record variable (for assignment, parameter, comparison, etc) just use its name
- To refer to a field of a record, use `record_name.field_name`
  - `average_male.weight`
  - `average_female.name`

3. Using records

- Assignment
  - You can assign one record variable to another of identical type
    - `that_person := this_person;`

- Input
  - You cannot read an entire record variable in a single operation. You must read each field separately.

  - To input a record variable use a procedure:
    - Prompt for and get each field in turn
CQ 1

1. My_First_Record contains contents of My_Second_Record

2. Program will not compile

3. Program gives a run-time error

4. Don’t know

3. Using records

- **Output**
  - You cannot display an entire record variable in a single operation. You must **display each field separately**.

  - To display a record variable use a procedure:
    - Describe and display each field in turn
3. Using records

- **Comparisons**
  - You can compare one record variable to another of identical type using "=" or "/=" operators
    - `if this_person = that_person then`
  - You should use a function to compare specific fields
    - `function is_heavier_than(a_person, another_person : persons) return BOOLEAN is`
      ```
      begin  -- is_heavier_than
        return a_person.weight > another_person.weight;
      end is_heavier_than;
      ```
    - To use this function:
      ```
      if is_heavier_than(this_person, that_person) then
        PUT(this_person.name); PUT_LINE(" is heavier.");
      else
        PUT(that_person.name); PUT_LINE(" is heavier.");
      end if;
      ```

CQ 2

1. Displays garbage
2. Program will not compile
3. Program gives a run-time error
4. Displays
   - John Doe
   - 25
   - Detroit Mi
5. Don’t know
Hierarchical records

- The components of a record can be any type, including another record

```
persons

name
  title
fname
sname
phone
weight
  age
  sex
```

text_io

- Text_IO
  - Page line character
    - set_col: go to nominated column in output file
    - new_line: go to next line of output
    - set_line: go to nominated line in output file
    - new_page: go to next page of output
    - skip_line: go to start of next line in input
    - skip_page: go to start of next page of inputs
    - page: what page number are we up to in the file?
    - line: what line number are we up to on the page?
    - col: what character position are we up to on the line?
example

- SET_LINE (2);
  SET_COL (30);
  PUT ("Student Results Report");
  SET_LINE (4);
- SET_COL (5); PUT ("Student name");
- SET_COL (35); PUT ("Assignments");
- SET_COL (50); PUT ("Exams");
- SET_COL (65); PUT ("Average");
  SET_LINE (6);

Line length

- For output files
  - set_line_length for lines
  - set_page_length for pages
- set_line_length
  - EOL generated automatically when limit reached
  - Default is 0

- SET_LINE_LENGTH (30);
  for i in 1 .. 20 loop
    PUT (i**2, width => 5);
  end loop;

    ', 1  4  9 16 25 36'
    ', 49 64 81 100 121 144'
    ', 169 196 225 256 289 324'
    ', 361 400'
Files

• Files need to be:
  – Declared
    • File variable set up
      Open (Inf, In_File, File_Name(1..Name_Length));
  – Created/opened/reset
    • Disk file linked to file variable
    • File opened for I/O
      Mode is (In_File, Out_File, Append_File);
  – Used for I/O
    • PUT, GET, etc
      Put_Line (Outf, Line (1..Line_Length));
  – Closed
    • After I/O finished

CQ 3

In the program, what is changed in the file

1. this is without putline –
   Where does this line go?

2. this is without putline - Where does this line go?

3. This is a copy - do not replicate this is without putline -
   Where does this line go?

4. None of the Above
reset

- Need to process a file twice. RESET procedure:
  - Go back to beginning
  - (optionally) change mode
  - File must be open already

-- read file twice
open (filevar, in_file, filename);
-- code to read from the file
reset (filevar);
-- code to read the file all over again
close (filevar);

File position functions

- END_OF_FILE
  - Next character is EOF
  - Next character is combination of EOL, EOP, EOF
- END_OF_LINE
  - Next character is EOL or EOF
- END_OF_PAGE
  - Next character is combination of EOL and EOP
  - Next character is EOF

if END_OF_PAGE (infile) then ...
while not END_OF_FILE loop ...
Example 2 Priority Queue

- Data structure that stores items so that retrieval of 'highest priority' item can be done efficiently.
- Highest priority have lower values
- Operations: PUT, GET, EMPTY