Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: B pp. 302-318, FK pp. 385-413, 469-474

Lecture 14
Oct 6 2003

Data Structures

- Program design = data structures + algorithms
  - Arrays, Stacks, Queues, Linked lists, Hash tables, Trees, Graphs, …
  - Binary search, insertion sort, …
- Static vs. dynamic data structures
- Linear data structures
  - The elements form a sequence or linear list

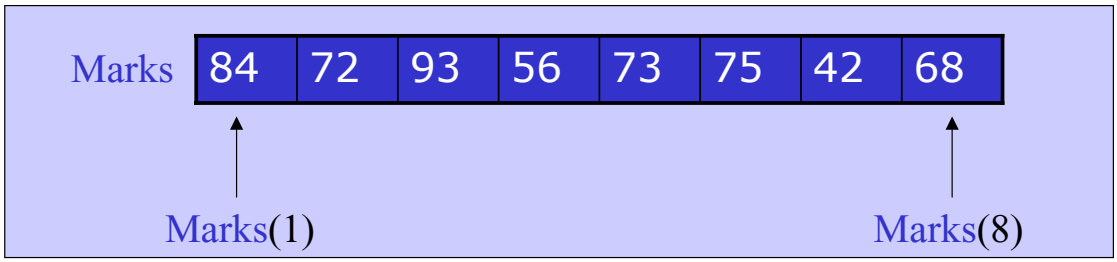
Linear lists

- Ordered collection of data
  - data is arranged into some order (not necessarily sorted)
  - data is referenced by its position within the list

Operation	Explanation
Initialize	Initialize the internal structure of the list, make sure it's empty
Empty	Returns true iff the list is empty
Insert	Inserts a new element after the k^{th} element; if k is zero, then insert at the beginning of the list
Delete	Delete the k^{th} item in the list

Arrays

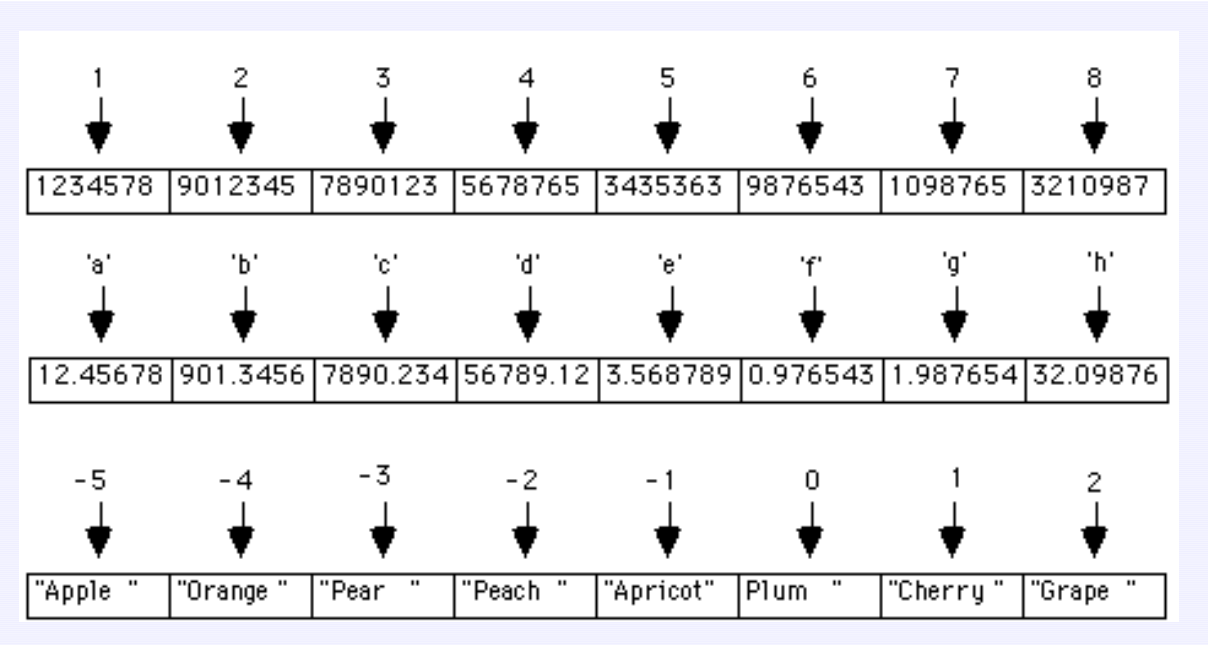
- *Data structure* which groups related items together
  - related in that they record similar data about several different things
    - the mark on a test for each student in a class
    - the temperature on the hour, at each hour during a day
    - etc



Array index vs. array element

- When designing an array, you need to decide
  - what the **labels** are going to be
    - the array *index*
    - what *type* of value is the index?
    - what *range* of values can the index take?
    - the array index may be INTEGER, CHARACTER or any ENUMERATED TYPE
  - what **type of information** can go into each box.
    - the array *element* type
    - the array element type can be *any* type
  - the type of the array index is not related to the type of the array items

Example [1/3]



Example [2/3]

- INTEGERS(1..8)
  - element type is INTEGER
  - index type is INTEGER
  - index can take 8 possible values, ranging from 1..8
- FLOATS('a'..'h')
  - element type is FLOAT
  - index type is CHARACTER
  - index can take 8 possible values, ranging from 'a'..'h'
- STRINGS(-5..2)
  - element type is STRING
  - index type is INTEGER
  - index can take 8 possible values, ranging from -5..2

Declaring Arrays

```
type Marks is array (1 .. 8) of integer;
X : Marks;
```

- An array declaration describes the *form* of the array
  - type of each element
    - can be anything
  - type and range of index
    - can be any ordinal type (INTEGER, CHARACTER, enumeration type, or any derived type or subtype of these)
  - element type **is not related** to index type

Example [3/3]

```
-- various constants used in data types

max_iarr : constant := 8;     -- largest index in int array
min_farr : constant := 'a';   -- low index in float array
max_farr : constant := 'h';   -- high index in float array

-- type declarations

subtype STRING8 is STRING (1 .. 8);

type int_8_array  is array (1 .. max_iarr) of INTEGER;
type float_arrays is array (min_farr..max_farr) of FLOAT;
type str_arrays   is array (-5 .. 2) of STRING8;
type small_arrays is array ('a' .. 'c') of FLOAT;
```

The declaration gives a name to the array type
then can declare variables of that array type

```
arr1 : int_8_array;
arr2 : float_arrays;
arr3 : str_arrays;
```

Initializing Arrays

```
type small_arrays is array ('a' .. 'c') of FLOAT;
```

- An array **aggregate** can be used to list initial values for items in an array variable
  - using positional notation
  - using explicit index references

```
-- init array coord1 using a positional list
coord1 : small_arrays := (1.2, 2.4, 3.6);

-- init array coord1 using explicit index references
coord2 : small_arrays := ('c'=>3.6, 'b'=>2.4, 'a'=>1.2);
```

-
```
-- init array coord1 using others
coord3 : small_arrays := ('b'=>5.2, others => 0.0);
```

Using Arrays

- **Referring to arrays**
  - To refer to an entire array just use the array variable name.
    - **Note:** refer to the array *variable*, not the array *type*
  - To refer to an individual element in the array: specify the array variable name and the index value for the element you want.

```
PUT(coord1('b'));

total := coord1('a') + coord1('b') + coord1('c');

PUT(arr3(-2));
```

Array Attributes

- Give info about the array type or array variable

Attribute	Meaning
`**first**	The value of the smallest index
`**last**	The value of the largest index
`**range**	The entire range or index values
`**length**	The number of items in the array

Array Attributes

```
max_iarr : constant := 8;    -- largest index in int array
min_farr : constant := 'a';  -- low index in float array
max_farr : constant := 'h';  -- high index in float array

subtype STRING8 is STRING (1 .. 8);

type int_8_array  is array (1 .. max_iarr) of INTEGER;
type float_arrays is array (min_farr..max_farr) of FLOAT;
type str_arrays   is array (-5 .. 2) of STRING8;
type small_arrays is array ('a' .. 'c') of FLOAT;

arr1 : int_8_array;
arr2 : float_arrays;
arr3 : str_arrays;

subtype lc_letter is CHARACTER range 'a' .. 'z';
type freq_table is array (lc_letters) of INTEGER;

count : freq_table := (others => 0); -- freq counts
```

Exercise

Array type / variable and attribute	Value
int_8array'**first**	
float_arrays'**last**	
str_arrays'**range**	
arr3'**length**	
small_arrays'**range**	
small_arrays'**length**	
freq_table'**range**	
count'**range**	

Array Attributes in Loops

- A useful application of array attributes is setting the bounds of loop control variables:

```
for t in count'range loop
   PUT(t);
   PUT(count(t), width=>11); NEW_LINE;
end loop;
```

  - This causes "t" to take each index value in turn for the array "count", *regardless* of the index type and range.

Operation on arrays

- Assignment
  - You can assign one entire array variable to another of the same type
    - `coord1 := coord2;`
- Comparison
  - You can compare one array variable to another of the same type
    - Compares item by item
    -
```
if (coord1 /= coord2) then
     PUT("They are different");
end if;
```

Operation on arrays

- Arrays as Parameters
  - You can use an array variable as an actual parameter to a procedure or function.
  - The amount of flexibility you have in doing so depends on how the formal parameter was declared in the subprogram:
    - if an *unconstrained array type* is used for the formal parameter, then **any** variable based on that type may be passed as an actual parameter.
    - if a constrained array type is used for the formal parameter, then **only** variables of **that** type may be passed as an actual parameter

Unconstrained Arrays

- We have only used **constrained array types** so far
  - the size of array was specified in type declaration, when the range of index values was specified
- Ada also provides **unconstrained array types**
  - element type is specified in type declaration
  - index type is specified in type declaration
  - range of index values (ie size) is **not** specified in type declaration
  - specify range of index values in *variable* declarations

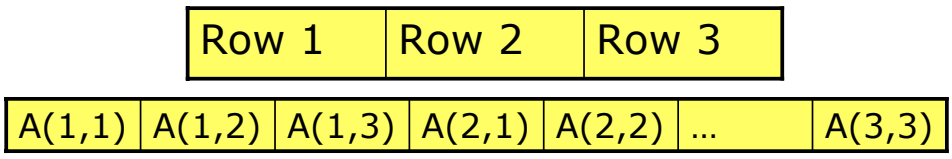
Representing 2D arrays as 1D arrays

- Row- and Column-major ordering

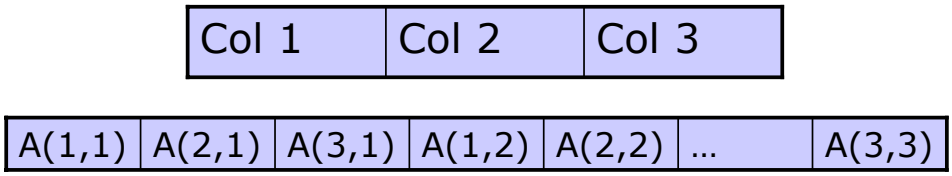
	Col 1	Col 2	…	Col n
Row 1	X	X	…	X
Row 2	X	X	…	X
…	…	…	…	…
Row m	X	X	…	X

Representing 2D arrays as 1D arrays

- Row-major



- Column-major

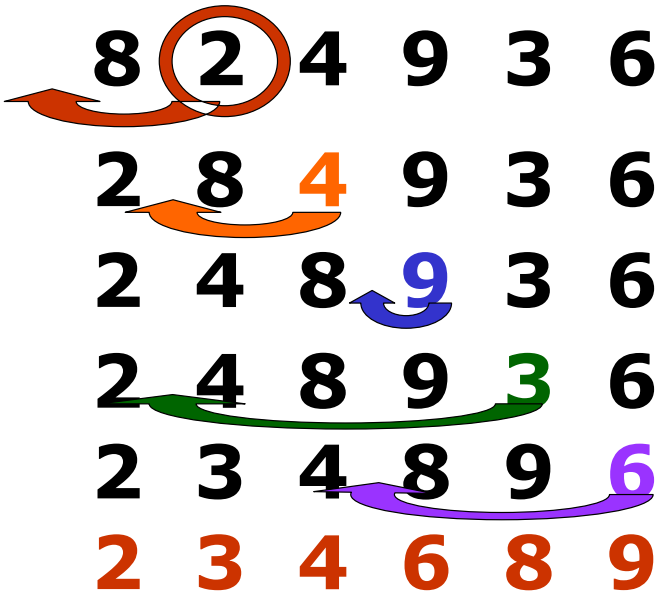


Insertion sort

- Uses a fixed amount of storage beyond what is needed for the data

- ```
  InsertionSort(A)
  -- A array of n numbers
    for j in 2 to length of A loop
      key := A(j)
      i   := j-1
      while i > 0 and A(i) > key
        A(i+1) := A(i)
        i      := i-1
        A(i+1) := key
  ```

Insertion sort


  ```

Bubble Sort

```
last := length;
for I in 1 .. Last -1 loop
   for J in I+1 .. Last loop
        if List(I) < List(J) then
             swap list(i) and list(j)
        end if
   end loop
end loop
```

Bubble Sort

11	**34**	26	90	37	58	10	47	36
34	**11**	**26**	90	37	58	10	47	36
34	26	**11**	**90**	37	58	10	47	36
34	26	90	**11**	**37**	58	10	47	36
34	26	90	37	**11**	**58**	10	47	36
34	26	90	37	58	**11**	**10**	47	36
34	26	90	37	58	11	**10**	**47**	36
34	26	90	37	58	11	47	**10**	**36**
34	26	90	37	58	11	47	36	**10**
34	90	37	58	26	47	36	11	10
90	37	58	34	47	36	26	11	10
90	58	37	47	36	34	26	11	10
90	**58**	**47**	**37**	**36**	**34**	**26**	**11**	**10**