Lecture C15: Matrices

Response to 'Muddiest Part of the Lecture Cards'

(38 respondents)

1. ***Still don't understand 'Length, 'Range operators on Arrays. How do the two concepts scale for multi-dimensional arrays? (5 students)***

For One-Dimensional arrays,

- Array_Name'Length: gives the number of elements that can be stored in the array (number of values that the index of the array can take).
- Array_Name'Range: specifies the range Array_Name'First .. Array_Name'Last for the single dimension.

  Note: Array_Name'Length = Array_Name'Last – Array_Name'First + 1

For multi-dimensional arrays,

- Array_Name'Length(N): denotes the number of values of the N-th index range (zero for a null range);
- Array_Name'Range(N): specifies the range Array_Name'First(N) .. Array_Name'Last(N) for dimension-N.

2. ***Questions on the Concept Questions (3 students)***

```
with Ada.Text_Io;
with Ada.Float_Text_Io ;
with Ada.Integer_Text_IO;
procedure Three_Dimension_Array is
  type Float_Array is array (Integer range <>, Integer range <>, Integer range <>) of
Float;
  function Display(
  Value : in     Float_Array )
   return Float is
Sum : Float :=0.0;
    N : integer;
  begin
N := Value'Length(1) * Value'Length(2)*Value'length(3);
    --
        Ada.Text_Io.Put(Integer'Image(N));
Ada.Text_Io.New_Line;
    Ada.Integer_Text_Io.Put(Value'Length(1) );
Ada.Text_Io.New_Line;

    Ada.Integer_Text_Io.Put(Value'Length(2) );
Ada.Text_Io.New_Line;
    Ada.Integer_Text_Io.Put(Value'Length(3) );
Ada.Text_Io.New_Line;
```

```
for I in Value'range(1) loop
        for J in Value'range(2) loop
        for K in Value'range(3) loop
                        Sum := Sum + Value(I,J,K);
                        Ada.Float_Text_Io.Put(Value (I,J,K));
                        Ada.Text_IO.Put(" , ");
        end loop;
        Ada.Text_IO.New_Line;
        end loop;
        Ada.Text_IO.New_Line;
end loop;
return (Sum /float(N));
end Display;
A : Float_Array := (((1.2, 2.3), (3.4, 4.5), (5.6, 7.8)),
                    ((1.2, 2.3), (3.4, 4.5), (5.6, 7.8)));

begin
        Ada.Float_Text_Io.Put (Display(A));
        Ada.Text_Io.New_Line;

end Three_Dimension_Array;
```

Note: **Value** here is the **name of the array**. It **does not** refer to type'Value.

a. How do you know the value of N?

The value of N is computed using the formula, N := Value'Length(1) * Value'Length(2) * Value'Length(3).

= 2 * 3 * 2 = 12

Value'Length returns an integer (size of the dimension). Hence the product is also an integer.

b. What are the values of Value'Range(1), Value'Range(2), Value'Range(3)?

Value'Range(1) = 1 .. 2
Value'Range(2) = 1 .. 3
Value'Range(3) = 1 .. 2

3. *What are the operations defined for matrices? (4 students)*

Take a look at the Demo_Gra.adb program that was handed out in class. In addition, read the specification files for Generic_Real_Arrays, Generic_Real_Arrays.Operations and Generic_Real_Array.Io.

4. *I still don't understand operations on arrays? (7 students)*

A Linear List is an ordered collection of data; that is, the data is arranged into some order (not necessarily sorted), so that data is referenced by its position within the list. Arrays provide a means of implementing linear lists. The basic operations on linear lists are summarized in table 1 below.

Operation	Explanation
Initialize	Initialize the internal structure of the list and ensure that it is empty
Empty	Boolean function that returns true if and only if (iff) the list is empty
Insert (K)	Subprogram to insert an element into the list at position k
Delete (Element)	Subprogram to delete an element from the list
Sort	Subprogram to sort the elements in the list

Table 1. Summary of Operations on Linear Lists

A linear list can be implemented using the Array construct in Ada95. Consider the package specification shown below:

```
-- Package specification of an array implementation of a linear list
package My_Array is

  My_Array_Max : constant Integer:=10;
  My_Array_Min : constant Integer:=1;
  type My_Integer_Array is array (My_Array_Min .. My_Array_Max) of Integer;

  -- procedure to create an array of My_Array_Max numbers, of type
My_Integer_Array
  procedure Create (Output_Array: in out My_Integer_Array);

  -- procedure to display the contents of an array
  procedure Display (Output_Array: in My_Integer_Array);

  -- function to determine if the array is empty
  -- The array is said to be empty if the elements in the array are all 0
  function Is_Empty(Output_Array: in My_Integer_Array) return Boolean;

  -- function to linearly scan an array. Returns the index if the number is found.
  -- returns -1 if no number was found.
  function Linear_Search (Input_Array: My_Integer_Array; Search_For: integer)
return Integer;


  -- procedure to delete an element Element in the array
  procedure Delete (Output_Array: in out My_Integer_Array; Element: in integer);

  -- procedure to insert an element into the array at position k
  procedure Insert (Output_Array: in out My_Integer_Array; index : in integer;
Element : in integer);


  -- procedure to bubble sort an array
```

```
   procedure Bubble_Sort (Sort_Array : in out My_Integer_Array);

end My_Array;


-- implementation of My_Array package
package body My_Array is
   procedure Create (
         Output_Array : in out My_Integer_Array ) is
      Number : Integer;
   begin
      for I in 1..My_Array_Max loop
         Put("Please enter the number : ");
         Get (Number);
         Skip_Line;
         Output_Array(I) := Number;
         New_Line;
      end loop;
   end Create;

   -- procedure to display the contents of Output_Array on the screen
   -- Accepts the output array and displays it to the user on screen
   procedure Display (
         Output_Array : in     My_Integer_Array ) is

   begin
      for I in 1..My_Array_Max loop
         Put (Output_Array(I));
         New_Line;
      end loop;
   end Display;

   -- procedure to linearly scan an array. Returns the index if the number is found.
   -- returns -1 if no number was found.
   function Linear_Search (
         Input_Array : My_Integer_Array;
         Search_For  : Integer          )
     return Integer is
      Index : Integer := - 1;

   begin
      for I in 1.. My_Array_Max loop
         if (Input_Array(I) = Search_For) then
            Index := I;
            exit;
         end if;
      end loop;
      return (Index);
   end Linear_Search;

   -- function that returns true if array is empty
   function Is_Empty (
         Input_Array : My_Array )
     return Boolean is
   begin
      for I in 1 .. My_Array_Max loop
```

```
      if Input_Array(I) /= 0 then
        return False;
      end if;
      return True;
    end loop;
end Is_Empty;



-- procedure to perform a simple bubble sort operation
-- accepts the array and returns the array sorted in descending order

procedure Bubble_Sort (
      Sort_Array : in out My_Integer_Array ) is
   Temp : Integer;
begin
   for I in 1 .. My_Array_Max loop
      for J in I+1 .. My_Array_Max loop
         if (Sort_Array(I) <= Sort_Array(J)) then
            Temp := Sort_Array(I);
            Sort_Array(I) := Sort_Array(J);
            Sort_Array(J) := Temp;
         end if;
      end loop;
   end loop;

end Bubble_Sort;

-- procedure to insert an element into the array at position 'index'
-- the subprogram assumes that there is atleast one empty spot in
-- the array. it copies the elements from my_array_max-1 .. index 1 step
-- and then inserts the element in its position
procedure Insert (
      Output_Array : in out My_Integer_Array;
      Index        : in     Integer;
      Element      : in     Integer          ) is
begin
   for I in reverse Index .. My_Array_Max -1 loop
      Output_Array(I+1) := Output_Array(I);
   end loop;
   Output_Array(Index):= Element;
end Insert;

-- procedure to delete 'Element' from the array.
-- the subprogram assumes that there is atleast one empty spot in
-- the array. it copies the elements from index+1 .. my_Array_Max one step
-- to the left and inserts a 0 at Output_Array(my_array_max)
procedure Insert (
      Output_Array : in out My_Integer_Array;
      Element      : in     Integer          ) is
   Index : Integer;
begin
   loop
      Index := Linear_Search(Output_Array, Element);
      exit when Index = -1;
      for I in Index+1 .. My_Array_Max  loop
```

```
            Output_Array(I-1) := Output_Array(I);
          end loop;
          Output_Array(My_Array_Max):= 0;
        end loop;

      end Delete;
    end My_Array;
```

When we are dealing with multidimensional arrays, the array is referenced by an index for each dimension. For example consider the 2-D array shown below in table 2:

1	2	3
4	5	6
7	8	9

Table 2. Simple 2-D array

If the array were called My_Table:

My_Table(1,1) = 1
My_Table(1,2) = 2
My_Table(1,3) = 3
My_Table(2,1) = 4
My_Table(2,2) = 5
My_Table(2,3) = 6
My_Table(3,1) = 7
My_Table(3,2) = 8
My_Table(3,3) = 9

If loops are used to control access to My_Table, then the loops

```
for I in 1 .. 3 loop
  for J in 1 .. 3 loop
    Ada.Integer_Text_Io.Put(My_Table(I,J));
    Ada.Text_Io.Put(" , ");
  end loop;
  Ada.Text_Io.New_Line;
end loop;
```

Try tracing the changing values of I, J and see what the final output will be at the end of the program.

The aggregation operation can be used for multidimensional arrays as well. In the distance program seen in lecture, we used an aggregation of the form

Traveled : distance_table := (others=> others => 0). The statement instructs the compiler to initialize all the elements in the array to 0.

5. ***No Mud? (17 students)***

Cool!