Exercise 1.1: Type equivalence

Given are the following declarations:

```plaintext
TYPE Array1 = ARRAY[1..10] OF BOOLEAN;
    Array2 = ARRAY[1..10] OF BOOLEAN;
VAR A : Array1;
    B, C : Array[0..9] OF BOOLEAN;
    D, E : Array2;
    F : Array2;
    X : ARRAY[1..10] OF Array1;
    Z, Y : ARRAY[2..11] OF Array2;
```

- Which variables are structurally equivalent?
- Which variables are name equivalent? (Some cases are treated differently in different languages. Which cases are those and what possibilities exist?)
- Name advantages and disadvantages of the two concepts of type equivalence!

Exercise 1.2: Type conversions

We have the following code snippet:

```plaintext
a, b: real;
c: int;

c := a + b;
```

Assume that the variables have the values `a = 3.6` and `b = 3.8`. What value does `c` have after the assignment in a language with implicit type conversion (from `real` to `int`) if

- the conversion is done by mathematical rounding and is done as early as possible,
- the conversion is done by mathematical rounding and is done as late as possible,
- the conversion is done by truncating decimal places and is done as early as possible,
- the conversion is done by truncating decimal places and is done as late as possible?
Exercise 1.3: Record and array layout

procedure Test (A: Integer; B: Integer) is
  I: Integer;
  type Rec is record
    Flags: array (1..3) of Boolean;
    Arr: array (1..A) of Integer;
    Arr2: array (1..B) of Integer;
    Status: Boolean;
    Count: Integer;
  end record;
  RObj: Rec;
  K:  Integer;
begin ... end Test;

• At what time is the size of the variable RObj determined?
• Describe the layout of the record RObj. You are allowed to use different representations for each of the arrays inside RObj.
• Describe the layout of the activation record (on the stack) in detail.

Exercise 1.4: Multi-dimensional array

The following Ada 95 code operates on 2-dimensional arrays of arbitrary size:

type Matrix is array (Integer range <>, Integer range <>) of Float;
type Matrix_Access is access all Matrix;

procedure Main is
  M1 : aliased Matrix (1 .. 10, 5 .. 17);
  M2 : Matrix_Access := new Matrix (1 .. 10, 5 .. 17);
  M3 : Matrix_Access := new Matrix (3 .. 6, 1 .. 2);
begin
  -- M3 := M1'Access; -- (1) let M3 point to M1
  -- M3 := M2; -- (2) let M3 point to M2
  ... := M3.all (5, 1);
end Main;

• Where are the contents of M1 stored in procedure Main? On the stack or on the heap? Where are the contents of M2 stored? What about M3?
• For the access to M3 the array dope is needed. Why? Where could it be stored? Is array dope required for any of the other two arrays?
• What would happen if the line marked with (1) were active and not commented out? Why? What if the line marked with (2) were active instead?
• Show the contents of the dope. Show how the elements of M3.all are arranged in memory, assuming row-major array layout. What would be the difference in column-major?