Solutions to Exercises 2

2.2.2 Represent sums of money in cents, as follows:

(a) C or C++: use type **`int`** or **`long`** (whichever is 32 bits).

(b) JAVA: use type **`int`**.

(c) ADA:

```
type Money is range 0 .. 10000000;
```

2.2.3 There are about 200 countries in the world. Represent them as follows:

(a) C or C++: use a suitable enumeration type.

(b) JAVA: use type **`char`** (unsigned 16 bits).

(c) ADA: use a suitable enumeration type.

2.3.1

(a) C++ types:

Suit = {*club*, *diamond*, *heart*, *spade*}	#Suit = 4
Card = Suit × Byte	#Card = 4 × 256 = 1024
Hand = {0, 1, …} → Card	#Hand = 1024^{2M}
Turn = Boolean × Card	#Turn = 2 × 1024 = 2048

(b) ADA types:

Suit = {*club*, *diamond*, *heart*, *spade*}	#Suit = 4
Rank = {2 , …, 14}	#Rank = 13
Card = Suit × Rank	#Card = 4 × 13 = 52
Hand = {1, …, 7} → Card	#Hand = 52^7
Turn = *false* Card + *true* Unit	#Turn = 52 + 1 = 53

2.3.2 C arrays are static; JAVA arrays are flexible (and are classified as objects); ADA arrays are dynamic. C and JAVA arrays have integer index ranges with lower bound 0; ADA arrays have index range of any discrete primitive type or subtype.

(a) Accessing `a1[i]` when `i` is out of range causes a C program to behave unpredictably; a JAVA or ADA program throws a suitable exception.

(b) Assigning `a2` to `a1` in C is illegal; in JAVA it makes `a1` contain a reference to the same array as `a2`; in ADA it assigns to `a1` a copy of the `a2` array.

2.3.3 Since a JAVA array is an object, it can be used wherever an object is allowed. For instance, the components of a `List` or `Set` may be arrays as well as other objects.

Also, reference semantics is adopted for arrays as for all other objects.

2.3.5 Arrays and function procedures both implement mappings. The fundamental difference is that an array implements the mapping by means of a data structure (whose index must be finite), whilst a function procedure does so by means of an algorithm (whose argument need not be finite).

(a) In ADA:

```
not: constant array (Boolean) of Boolean
       := (false => true, true => false);
```

```
function not (b: Boolean) return Boolean is
begin
    if b then
      return false;
    else
      return true;
    end if;
end;
```

(b) In ADA:

```
type Small is Natural range 0 .. 10;
fac: constant array (Small) of Positive
      := (0 => 1, 1 => 1, 2 => 2, 3 => 6,
          4 => 24, 5 => 120, 6 => 720,
          7 => 5040, 8 => 40320,
          9 => 362880, 10 => 3628800);
function fac (n: Small) return Positive is
begin
    if n <= 1 then
      return 1;
    else
      return n * fac(n-1);
    end if;
end;
```

* **2.6.2** Possible extension to ADA:

```
Expression  ::=  …
              |  if Expression
                 then Expression
                 else Expression end if ;
```

** **2.6.4** Possible extension to JAVA (array constructions):

```
Expression  ::=  …
              |  { Expression ( , Expression )* }
```

2.7.1 Representation of ADA types of Exercise 2.3.1:

The enumeration type `Suit` would be represented by a single cell containing one of {*club*, *diamond*, *heart*, *spade*} (where the enumerands are represented by 0, 1, 2, and 3, respectively).

The integer type `Rank` would be represented by a single cell containing one of {2, …, 14}.

The record type `Card` would be represented by a pair of fields, where the first field contains a `Suit` value and the second field contains a `Rank` value. (See illustration below.)

The array type `Hand` would be represented by a row of 7 pairs of fields, where each pair of fields represents a `Card` record. (See illustration below.)

The discriminated record type `Turn` would be represented by a tag field containing a boolean value, together with *either* a pair of fields representing a `Card` record (if the tag is *false*) *either* nothing (if the tag is *true*). (See illustration below.)