Exercises

Exercise 1

Given the following package definition:

```
package Expr_Eval is

    type Expr;

    type Expr_Kind is (Bin_Op, Literal, If_Expr);
    type Op_Kind is (Add, Sub, Mul, Div, Logic_And, Logic_Or);
    type Expr_Access is access Expr;

    type Expr (Kind : Expr_Kind) is record
      case Kind is
         when Bin_Op =>
            L, R : Expr_Access;
            Op   : Op_Kind;
         when If_Expr =>
            Cond, Then_Expr, Else_Expr : Expr_Access;
         when Literal =>
            Val : Integer;
      end case;
    end record;

    function Eval (E: Expr) return Integer;

end Expr_Eval;
```

Complete it with a body.

Here is a test (write more than this one).

```
with Expr_Eval; use Expr_Eval;

procedure Test is
    E : Expr := (Kind => Bin_Op,
                 L => new Expr'(Kind => Literal, Val => 12),
                 R => new Expr'(Kind => Literal, Val => 15),
                 Op => Add)
begin
    Put_Line (Eval (E)'Image);
end Test;
```

Exercise 2

Transform exercise one to use `Indefinite_Holders` instead of an access type. (See http://www.ada-auth.org/standards/12rat/html/Rat12-8-5.html)

Exercise 3

Transform exercise two to use a tagged type hierarchy instead of a discriminated record.

```
package Expr_Eval is

    type Op_Kind is (Add, Sub, Mul, Div, Logic_And, Logic_Or);
    type Expr_Access is access Expr;

    type Expr is tagged null record;

    type Bin_Op is new Expr with record
    end record;

    type If_Expr is new Expr with record
    end record;

    type Literal is new Expr with record
    end record;

    function Eval (E: Expr) return Integer is abstract;

    overriding function Eval (B : Bin_Op) return Integer;

    ...

    --  Add all the overrides of for derived classes

end Expr_Eval;
```

Exercise 4

Extend your prefered version to handle two more expression kinds:

- `Let`. The let expression allows the user to introduce a temporary binding from a name to a value.
- `Ref`. Ref allows referencing a name, introduced by a let, and the result of the evaluation will be the value of the binding.

  To represent the scopes, you can use either an array, or a hash map. Hash maps are in `Ada.Containers.Hashed_Maps`

Exercise 5 [BONUS]

Add a type check function, and a boolean literal. Your type check function must verify that boolean operators are only used on booleans, that the if expression's

condition is of a boolean type, and that arithmetic operators are only used on integers.