Exercise 5: Task 4

C Implementation of MurmurHash2 in Ada

What is a Cryptographic Hash Function?

- mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size
- designed to be a one-way function (infeasible to invert)
- MurMurHash created by Austin Appleby in 2008
- name comes from two basic operations, multiply (MU) and rotate (R)
- not specifically designed to be difficult to reverse

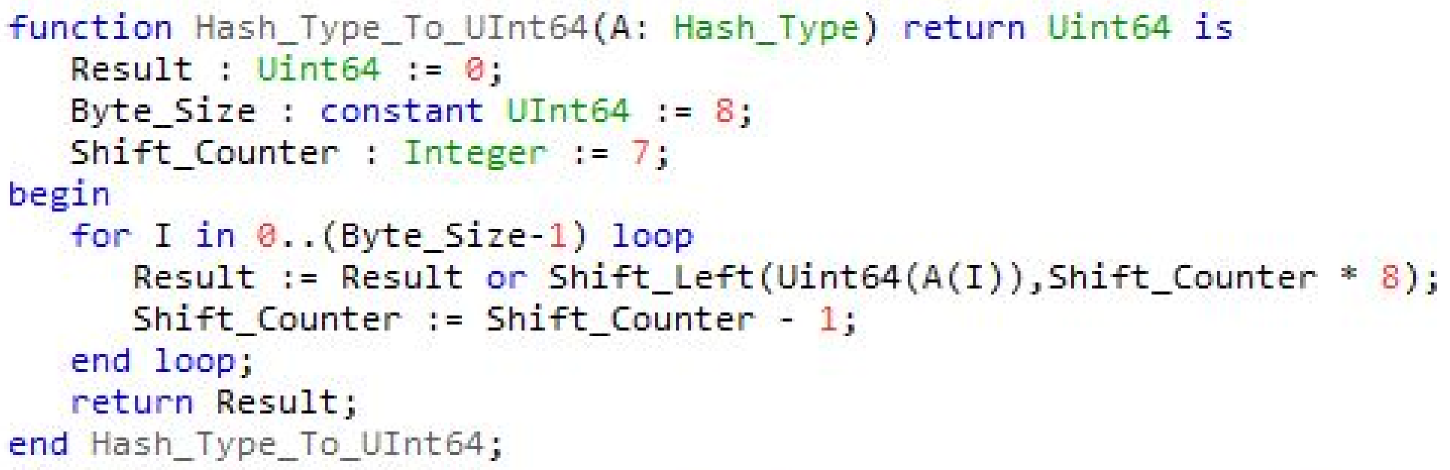
Challenges:

- Presentation of Bits and Bytes with operations
- Little-Endian and Big-Endian
- “C” Switch to “Ada” Cases Implementation

Presentation of Bits and Bytes with operations

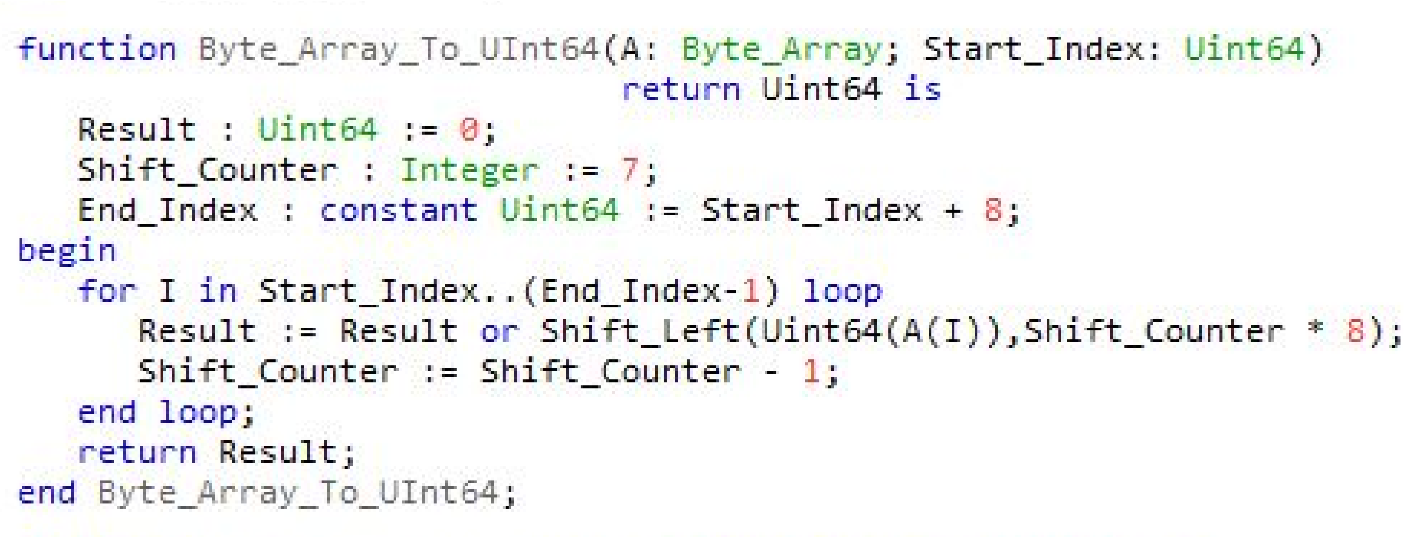
- **use array with 8 x Uint8**
  - use simply format of template (.ads), xor and shift easy to implement
  - implement multiplication on array
- **use UInt64**
  - all operations are given by “Interfaces”
  - needs 2 functions to transform between 8 Byte-Array ↔ UInt64
- **use array 64 x Boolean**
  - some operations are given (xor, shift)
  - implementation of transform functions and multiplication

Presentation of Bits and Bytes with operations



```
function Hash_Type_To_UInt64(A: Hash_Type) return Uint64 is
   Result : Uint64 := 0;
   Byte_Size : constant UInt64 := 8;
   Shift_Counter : Integer := 7;
begin
   for I in 0..(Byte_Size-1) loop
      Result := Result or Shift_Left(Uint64(A(I)),Shift_Counter * 8);
      Shift_Counter := Shift_Counter - 1;
   end loop;
   return Result;
end Hash_Type_To_UInt64;
```

Presentation of Bits and Bytes with operations



```
function Byte_Array_To_UInt64(A: Byte_Array; Start_Index: Uint64)
                              return Uint64 is
   Result : Uint64 := 0;
   Shift_Counter : Integer := 7;
   End_Index : constant Uint64 := Start_Index + 8;
begin
   for I in Start_Index..(End_Index-1) loop
      Result := Result or Shift_Left(Uint64(A(I)),Shift_Counter * 8);
      Shift_Counter := Shift_Counter - 1;
   end loop;
   return Result;
end Byte_Array_To_UInt64;
```

Little-Endian and Big-Endian

- Numbers can have a different presentation (e.g. 1025)
  - **Little-Endian:** 00000001 00000100 00000000 00000000
  - **Big-Endian**: 00000000 00000000 00000100 00000001

- used Big-Endian in Hash-Function
  - better readable
  - more logical transformation between 8 Byte-Array ↔ UInt64

“C” Switch to “Ada” Cases Implementation

C:

```
switch (len ) {
        case 7: h ^= ( uint64_t )( data [6]) << 48;
        case 6: h ^= ( uint64_t )( data [5]) << 40;
        case 5: h ^= ( uint64_t )( data [4]) << 32;
        case 4: h ^= ( uint64_t )( data [3]) << 24;
        case 3: h ^= ( uint64_t )( data [2]) << 16;
        case 2: h ^= ( uint64_t )( data [1]) << 8;
        case 1: h ^= ( uint64_t )( data [0]);
        h *= m;
};
```

Ada?

```
case Len is
      when 7 => h := h xor Shift_Left(Message(6), 48);
      when 6 => h := h xor Shift_Left(Message(5), 40);
      when 5 => h := h xor Shift_Left(Message(4), 32);
      when 4 => h := h xor Shift_Left(Message(3), 24);
      when 3 => h := h xor Shift_Left(Message(2), 16);
      when 2 => h := h xor Shift_Left(Message(1), 8);
      when 1 => h := h xor Message(0);
      when others => h := h * m;
    end case;
```

“C” Switch to “Ada” Cases Implementation

C:

```
switch (len ) {
        case 7: h ^= ( uint64_t )( data [6]) << 48;
        case 6: h ^= ( uint64_t )( data [5]) << 40;
        case 5: h ^= ( uint64_t )( data [4]) << 32;
        case 4: h ^= ( uint64_t )( data [3]) << 24;
        case 3: h ^= ( uint64_t )( data [2]) << 16;
        case 2: h ^= ( uint64_t )( data [1]) << 8;
        case 1: h ^= ( uint64_t )( data [0]);
        h *= m;
};
```

Ada?

```
case Len is
      when 7 => h := h xor Shift_Left(data(6), 48);
      when 6 => h := h xor Shift_Left(data(5), 40);
      when 5 => h := h xor Shift_Left(data(4), 32);
      when 4 => h := h xor Shift_Left(data(3), 24);
      when 3 => h := h xor Shift_Left(data(2), 16);
      when 2 => h := h xor Shift_Left(data(1), 8);
      when 1 => h := h xor data(0);
      when others => h := h * m;
    end case;
```

Not the same behavior! → Ada uses break after each case

The C and Ada Implementation in Comparison

→ see Code