VHDL

VHDL standard libraries

Unité de conception

- Unités primaires
  - l'entité (entity)
  - la configuration
  - le paquetage
- Unités secondaires
  - l 'architecture
  - le corps de paquetage

L’entité

- Définit les interfaces d’entrées-sorties d’une unité, c’est la vue externe

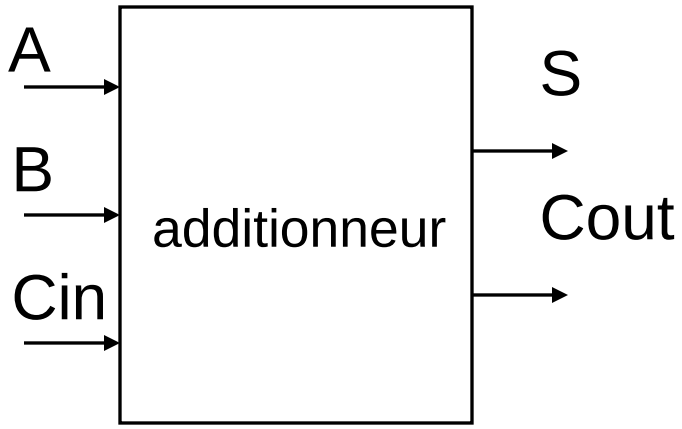
entity *additionneur* **is**

port(

A, B, Cin **: in** bit **;**

S, Cout **: out** bit**);**

end *additionneur* **;**



Paquetage

```
PACKAGE std_logic_1164 IS

    ------------------------------------------------------------------
    -- logic state system  (unresolved)
    ------------------------------------------------------------------
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                         '-'   -- Don't care
                       );
    ------------------------------------------------------------------
    -- unconstrained array of std_ulogic for use with the resolution function
    ------------------------------------------------------------------
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

```

```
------------------------------------------------------------------
   -- resolution function
   ------------------------------------------------------------------
   FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

   ------------------------------------------------------------------
   -- *** industry standard logic type ***
   ------------------------------------------------------------------
   SUBTYPE std_logic IS resolved std_ulogic;

   ------------------------------------------------------------------
   -- unconstrained array of std_logic for use in declaring signal arrays
   ------------------------------------------------------------------
   TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;

   ------------------------------------------------------------------
   -- common subtypes
   ------------------------------------------------------------------
   SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
   SUBTYPE X01Z    IS resolved std_ulogic RANGE 'X' TO 'Z'; -- ('X','0','1','Z')
   SUBTYPE UX01    IS resolved std_ulogic RANGE 'U' TO '1'; -- ('U','X','0','1')
   SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z'; -- ('U','X','0','1','Z')
```

overloaded logical operators

```
FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "xnor" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;
```

vectorized overloaded logical operators

```
FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "xnor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "xnor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "not"  ( l : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector;
```

conversion functions

```
FUNCTION To_bit       ( s : std_ulogic;        xmap : BIT := '0') RETURN BIT;
FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;

FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic;
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector;
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;
```

strength strippers and type convertors

```
FUNCTION To_X01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( s : std_ulogic        ) RETURN  X01;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( b : BIT               ) RETURN  X01;

FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( s : std_ulogic        ) RETURN  X01Z;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( b : BIT               ) RETURN  X01Z;

FUNCTION To_UX01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( s : std_ulogic        ) RETURN  UX01;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( b : BIT               ) RETURN  UX01;
```

edge detection

```
FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
```

Corps de paquetage IEEE1164

```
PACKAGE BODY std_logic_1164 IS
    -------------------------------------------------------------------
    -- local types
    -------------------------------------------------------------------
    TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
    TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

    -------------------------------------------------------------------
    -- resolution function
    -------------------------------------------------------------------
    CONSTANT resolution_table : stdlogic_table := (
    --      ---------------------------------------------------------
    --      |  U    X    0    1    Z    W    L    H    -        |   |
    --      ---------------------------------------------------------
            ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
            ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
            ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
            ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
            ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
            ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
            ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
            ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
            ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
        );
```

Résolution de fonctions

```
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
    VARIABLE result : std_ulogic := 'Z';  -- weakest state default
 BEGIN
    -- the test for a single driver is essential otherwise the
    -- loop would return 'X' for a single driver of '-' and that
    -- would conflict with the value of a single driver unresolved
    -- signal.
    IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
    ELSE
        FOR i IN s'RANGE LOOP
            result := resolution_table(result, s(i));
        END LOOP;
    END IF;
    RETURN result;
 END resolved;
```

tables for logical operations

truth table for "and" function

```
CONSTANT and_table : stdlogic_table := (
 --      ---------------------------------------------------
 --      |  U    X    0    1    Z    W    L    H    -      |   |
 --      ---------------------------------------------------
         ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
         ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
         ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
         ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
         ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
         ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
         ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
         ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
         ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
 );
```

truth table for "not" function

```
CONSTANT not_table: stdlogic_1d :=
--  ---------------------------------------------
--  |   U    X    0    1    Z    W    L    H    -   |
--  ---------------------------------------------
     ( 'U', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );
```

overloaded logical operators (with optimizing hints)

```
FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (and_table(l, r));
END "and";

FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN  (not_table ( and_table(l, r)));
END "nand";

FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (or_table(l, r));
END "or";
```

And
std_logic_vector

```
FUNCTION "and"  ( l,r : std_logic_vector ) RETURN std_logic_vector IS
    ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
    ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
    VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
    IF ( l'LENGTH /= r'LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'and' operator are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result'RANGE LOOP
            result(i) := and_table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
END "and";
```

And
std_ulogic_vector

```
FUNCTION "and"  ( l,r : std_ulogic_vector ) RETURN std_ulogic_vector IS
       ALIAS lv : std_ulogic_vector ( 1 TO l'LENGTH ) IS l;
       ALIAS rv : std_ulogic_vector ( 1 TO r'LENGTH ) IS r;
       VARIABLE result : std_ulogic_vector ( 1 TO l'LENGTH );
   BEGIN
       IF ( l'LENGTH /= r'LENGTH ) THEN
           ASSERT FALSE
           REPORT "arguments of overloaded 'and' operator are not of the same length"
           SEVERITY FAILURE;
       ELSE
           FOR i IN result'RANGE LOOP
               result(i) := and_table (lv(i), rv(i));
           END LOOP;
       END IF;
       RETURN result;
   END "and";
```

conversion functions

```
    FUNCTION To_bit ( s : std_ulogic; xmap : BIT := '0') RETURN BIT IS

        VARIABLE result : BIT;
    BEGIN
        CASE s IS
--            WHEN '0' | 'L' => result := '0';
--            WHEN '1' | 'H' => result := '1';
            WHEN '0'  => result := '0';
            WHEN '1'  => result := '1';
            WHEN OTHERS =>    result := xmap;
        END CASE;
            RETURN result;
    END;
    ------------------------------------------------------------------
    FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR IS
        ALIAS sv : std_logic_vector ( s'LENGTH-1 DOWNTO 0 ) IS s;
        VARIABLE result : BIT_VECTOR ( s'LENGTH-1 DOWNTO 0 );
    BEGIN
        FOR i IN result'RANGE LOOP
            CASE sv(i) IS
                WHEN '0' | 'L' => result(i) := '0';
                WHEN '1' | 'H' => result(i) := '1';
                WHEN OTHERS => result(i) := xmap;
            END CASE;
        END LOOP;
        RETURN result;
    END;
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        ' ',  -- 'U'
                        ' ',  -- 'X'
                        ' ',  -- '0'
                        ' ',  -- '1'
                        ' ',  -- 'Z'
                        ' ',  -- 'W'
                        ' ',  -- 'L'
                        ' ',  -- 'H'
                        ' '   -- '-'
                       );
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        '0',  -- 'U'
                        ' ',  -- 'X'
                        ' ',  -- '0'
                        ' ',  -- '1'
                        ' ',  -- 'Z'
                        ' ',  -- 'W'
                        ' ',  -- 'L'
                        ' ',  -- 'H'
                        ' '   -- '-'
                       );
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        '0',  -- 'U'
                        '0',  -- 'X'
                        ' ',  -- '0'
                        ' ',  -- '1'
                        ' ',  -- 'Z'
                        ' ',  -- 'W'
                        ' ',  -- 'L'
                        ' ',  -- 'H'
                        ' '   -- '-'
                       );
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        '0',  -- 'U'
                        '0',  -- 'X'
                        '0',  -- '0'
                        ' ',  -- '1'
                        ' ',  -- 'Z'
                        ' ',  -- 'W'
                        ' ',  -- 'L'
                        ' ',  -- 'H'
                        ' '   -- '-'
                       );
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        '0',  -- 'U'
                        '0',  -- 'X'
                        '0',  -- '0'
                        '1',  -- '1'
                        ' ',  -- 'Z'
                        ' ',  -- 'W'
                        ' ',  -- 'L'
                        ' ',  -- 'H'
                        ' '   -- '-'
                       );
```

conversion tables
example ToBit

```
TYPE logic_ToBit_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF BIT;
-------------------------------------------------------

CONSTANT cvt_to_Bit : logic_ToBit_table := (
                        '0',  -- 'U'
                        '0',  -- 'X'
                        '0',  -- '0'
                        '1',  -- '1'
                        '0',  -- 'Z'
                        '0',  -- 'W'
                        '0',  -- 'L'
                        '0',  -- 'H'
                        '0'   -- '-'
                       );
```

strength strippers and type convertors
to_x01

```
FUNCTION To_Bit  ( s : std_ulogic) RETURN BIT IS
  BEGIN
      RETURN (cvt_to_Bit(s));
  END;
```

conversion tables

```
 TYPE logic_x01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01;
TYPE logic_x01z_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF X01Z;
TYPE logic_ux01_table IS ARRAY (std_ulogic'LOW TO std_ulogic'HIGH) OF UX01;
------------------------------------------------------------
-- table name : cvt_to_x01
--
-- parameters :
--        in  :  std_ulogic  -- some logic value
-- returns    :  x01         -- state value of logic value
-- purpose    :  to convert state-strength to state only
--
-- example    : if (cvt_to_x01 (input_signal) = '1' ) then ...
--
------------------------------------------------------------
CONSTANT cvt_to_x01 : logic_x01_table := (
                     'X',  -- 'U'
                     'X',  -- 'X'
                     '0',  -- '0'
                     '1',  -- '1'
                     'X',  -- 'Z'
                     'X',  -- 'W'
                     '0',  -- 'L'
                     '1',  -- 'H'
                     'X'   -- '-'
                    );
```

strength strippers and type convertors to_x01

```
FUNCTION To_X01  ( s : std_logic_vector ) RETURN  std_logic_vector IS
    ALIAS sv : std_logic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_logic_vector ( 1 TO s'LENGTH );
BEGIN
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01 (sv(i));
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------------------
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector IS
    ALIAS sv : std_ulogic_vector ( 1 TO s'LENGTH ) IS s;
    VARIABLE result : std_ulogic_vector ( 1 TO s'LENGTH );
BEGIN
    FOR i IN result'RANGE LOOP
        result(i) := cvt_to_x01 (sv(i));
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------------------
FUNCTION To_X01  ( s : std_ulogic ) RETURN  X01 IS
BEGIN
    RETURN (cvt_to_x01(s));
END;
```

object contains an unknown

```
FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN IS
     VARIABLE result : BOOLEAN;
BEGIN
     result := FALSE;
    FOR i IN s'RANGE LOOP
        CASE s(i) IS
            WHEN 'U' | 'X' | 'Z' | 'W' | '-' => result := TRUE;
            WHEN OTHERS => NULL;
        END CASE;
    END LOOP;
    RETURN result;
END;
```

```
-------------------------------------------------------------------
    FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN IS
          VARIABLE result : BOOLEAN;
    BEGIN
          result := FALSE;
        FOR i IN s'RANGE LOOP
            CASE s(i) IS
                WHEN 'U' | 'X' | 'Z' | 'W' | '-' => result := TRUE;
                WHEN OTHERS => NULL;
            END CASE;
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------------
    FUNCTION Is_X ( s : std_ulogic ) RETURN  BOOLEAN IS
          VARIABLE result : BOOLEAN;
    BEGIN
          result := FALSE;
        CASE s IS
            WHEN 'U' | 'X' | 'Z' | 'W' | '-' => result := TRUE;
            WHEN OTHERS => NULL;
        END CASE;
        RETURN result;
    END;
```

```
-------------------------------------------------------------------
    FUNCTION To_X01  ( b : BIT_VECTOR ) RETURN  std_logic_vector IS
        ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
        VARIABLE result : std_logic_vector ( 1 TO b'LENGTH );
    BEGIN
        FOR i IN result'RANGE LOOP
            CASE bv(i) IS
                WHEN '0' => result(i) := '0';
                WHEN '1' => result(i) := '1';
            END CASE;
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------------
    FUNCTION To_X01  ( b : BIT_VECTOR ) RETURN  std_ulogic_vector IS
        ALIAS bv : BIT_VECTOR ( 1 TO b'LENGTH ) IS b;
        VARIABLE result : std_ulogic_vector ( 1 TO b'LENGTH );
    BEGIN
        FOR i IN result'RANGE LOOP
            CASE bv(i) IS
                WHEN '0' => result(i) := '0';
                WHEN '1' => result(i) := '1';
            END CASE;
        END LOOP;
        RETURN result;
    END;
```

BIT To_Xo1

```
FUNCTION To_X01  ( b : BIT ) RETURN  X01 IS
     VARIABLE result : X01;
BEGIN
        CASE b IS
            WHEN '0' => result := '0';
            WHEN '1' => result := '1';
        END CASE;
             RETURN result;
END;
```

package STANDARD is

```
-- Predefined enumeration types

type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);

-- Predefined numeric types
type integer is range -2147483648 to 2147483647;
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;

-- Real Numbers
type real is range -1.0E308 to 1.0E308;

-- Time / delay_length
type time is range -2147483647 to 2147483647
        units
                    fs;
                    ps = 1000 fs;
                    ns = 1000 ps;
                    us = 1000 ns;
                    ms = 1000 us;
                    sec = 1000 ms;
                    min = 60 sec;
                    hr = 60 min;
        end units;
```

package STANDARD is

```
type character is (
nul, soh, stx, etx, eot, enq, ack, bel,  bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
dle, dc1, dc2, dc3, dc4, nak, syn, etb, can,  em, sub, esc, fsp, gsp, rsp, usp,

' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-', '.', '/',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?',
'@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', del,

c128, c129, c130, c131, c132, c133, c134, c135, c136, c137, c138, c139, c140, c141, c142, c143,
c144, c145, c146, c147, c148, c149, c150, c151, c152, c153, c154, c155, c156, c157, c158, c159,

-- the character code for 160 is there (NBSP), but prints as no char

' ', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',  '', '®', '¯',
'°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼', '½', '¾', '¿',
'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï',
'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß',
'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ' );

-- String
type string is array (positive range <>) of character;
```