



Statements

Presented by Quentin Ochem

university.adacore.com

Some General Information

 Statements can be spread on several lines, and need to be completed by a;

```
Put_Line
(
"Hello World"
);
```

 Statements need to be written in a sequence of statement

```
procedure Main is

   Put_Line ("Hello World");
begin

   Put_Line ("Hello World");
end Main;
```

If Statements / If Expressions

If statements

```
if A = 0 then
    Put_Line ("A is 0");
elsif B = 0 then
    Put_Line ("B is 0");
else
    Put_Line ("Else...");
end if;
Ada

if (A == 0) {
    printf ("A is 0");
    printf ("B is 0");
    printf ("B is 0");
}

else {
    put_Line ("Else...");
}
```

If expressions

Condition Operators

Comparison



• Binary Boolean Operators and Short Circuit Forms

Binary Boolean Operators			Short Circuit Control Forms	
or	and	xor	or else	and then

A Note on the Binary Boolean Operators

 "and", "or" are not short-circuit, both operands are always evaluated

```
if X \neq 0 and Y \neq X > 1 then -- MAY RAISE AN EXCEPTION
```

 Short-circuit execution is used for "and then" and "or else"

```
if X \neq 0 and then Y \neq X > 1 then -- OK
```

Case Statement / Case Expression

```
case A is
    when 0 =>
        Put_Line ("zero");
    when -9 .. -1 | 1 .. 9 =>
        Put_Line ("digit");
    when others =>
        Put_Line ("other")
end case;
Ada

switch (A) {
    case 0:
        printf ("0");
    break;
    case -9:case -8:
    case -5:case -4:
    case -1:case 1:case 1:case 4:case 5:case 6:case 8:case 9:
        printf ("digital break;
    default:
        printf ("other")
```

```
switch (A) {
    case 0:
        printf ("0");
        break;
    case -9:case -8:case -7:case -6:
    case -5:case -4:case -3:case -2:
    case -1:case 1:case 2:case 3:
    case 4:case 5:case 6:case 7:
    case 8:case 9:
        printf ("digit");
        break;
    default:
        printf ("other");
}
```

```
Put_Line (
    (case A is
    when 0 =>
        "zero"
    when -9 .. -1 | 1 .. 9 =>
        "digit"
    when others =>
        "other"));
```

Case Statements Rules

 All values covered by the type of the expression should be covered

```
V : Integer := ...;
begin
    case V is
        when 0 =>
            Put_Line (0);
    end case; -- NOK!
```

Values must be unique

```
V : Integer := ...;
begin
    case V is
        when 0 =>
            Put_Line ("0");
        when Integer'First .. 0 => -- NOK!
            Put_Line ("Negative");
        when others =>
            null;
end case;
```

Loop Statement

Loop is introduced by the loop / end loop block

```
Put_Line ("Hello");

delay 1.0;
end loop;
```

Loop can be controlled by a exit condition through a while statement

```
while A < 10 loop
    Put_Line ("Hello");
    A := A + 1;
end loop;</pre>
```

Exit Statements

At any time, a loop can be broken by an exit statement

```
loop
    A := A + 1;

if A > 10 then
    exit;
end if;

B := B + 1;
end loop;
```

```
loop
    A := A + 1;
    exit when A > 10;
    B := B + 1;
end loop;
```

In case of nested loops, names can be given to exit an outer loop

```
Loop_1 : loop
    A := A + 1;

Loop_2 : loop
    B := B + 1;
    exit Loop_1 when B > 10;
end loop;
end loop;
```

For .. Loop

Iteration can be done over a range

```
for X in 1 .. 10 loop
    Put_Line ("Hello");
end loop;
```

The range is always given in ascending order

```
for X in 10 .. 0 loop
    Put_Line ("Hello"); -- not called
end loop;
```

 The iteration always goes to the successor, or predecessor if reverse is specified

```
for X in reverse 1 .. 10 loop
    Put_Line ("Hello");
end loop;
```

For .. Loop Control Variable

 The variable declared in the for loop can be explicitly typed

```
for X in Integer range 1 .. 10 loop
   Put_Line ("Hello");
end loop;
```

• It may be of any discrete type. When no range is specified, the whole type range is taken

```
for X in Character loop
   Put_Line (X);
end loop;
```

The control variable is constant in the loop

```
for X in 1 .. 10 loop
    X := X + 1;
end loop;
```

For .. Loop Range Evaluation



• The loop range is evaluated once, when entering the loop

```
for X in A .. B loop

B := B + 1; -- no effect on the loop
end loop;
```

Declare Blocks

 In any sequence of statements, it may be useful to reopen declarative blocks or a sub sequence of expressions (for new variable declaration, exception handlers...)

Can be done through a new declare bock

```
declare
   A : Integer;
begin
   A := 0;
end;
```

Null Statements

• It is illegal to have an empty sequence of statements but ...

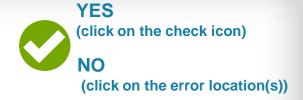
• ... it's possible to use a statement that does nothing, the null statement

```
declare
   A : Integer;
begin
   null;
end;
```





Is this correct? (1/10)



```
if A == 0 then
   Put_Line ("A is 0");
end if;
```

Is this correct? (1/10)



```
if A == 0 then
   Put Line ("A is 0");
end if;
```

Compilation error, = is the equality symbol in Ada

Is this correct? (2/10)



```
if A := 0 then
  Put Line ("A has been assigned to 0");
end if;
```

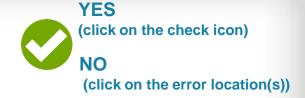
Is this correct? (2/10)



```
if A := 0 then
   Put Line ("A has been assigned to 0");
end if;
```

Compilation error, := is not an operator, can't be used in a condition

Is this correct? (3/10)



```
A: Integer := Integer'Value (Get_Line);
begin
    case A is
        when 1 .. 9 =>
            Put_Line ("Simple digit");
    when 10 .. Integer'Last =>
            Put_Line ("Long positive");
    when Integer'First .. -1 =>
            Put_Line ("Negative");
end case;
```

8

Is this correct? (3/10)



```
A: Integer := Integer'Value (Get_Line);

begin

case A is

when 1 .. 9 =>

Put_Line ('Simple digit");

when 10 .. Integer'Last =>

Put_Line ('Long positive");

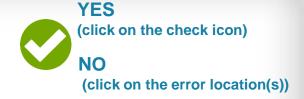
when Integer'First .. -1 =>

Put_Line ("Negative");

end case;
```

Compilation error, 0 value is missing

Is this correct? (4/10)



```
A : Integer := Integer'Value (Get Line);
begin
   case A is
      when 1 .. Integer'Last=>
         Put Line ("Positive");
      when 0 .. Integer'Last =>
         Put Line ("Natural");
      when others =>
         Put Line ("Other");
   end case;
```

Is this correct? (4/10)



```
A : Integer := Integer'Value (Get_Line);

begin

case A is

when 1 .. Integer'Last=>

Put_Line ("Positive");

when 0 .. Integer'Last=>

Put_Line ("Natural");

when others =>

Put_Line ("Other");

end case;
```

Compilation Error, 1 .. Integer'Last and 0 .. Integer'Last overlaps

Is this correct? (5/10)



```
A : Float := 10.0;
begin
    case A is
    when 1.0 .. Float'Last =>
        Put_Line ("Positive");
    when Float'First .. -1.0 =>
        Put_Line ("Negative");
    when others =>
        Put_Line ("Other");
    end case;
```



Is this correct? (5/10)



Compilation Error, the value evaluated must be discrete

```
A : Float := 10.0;

begin

case A is

when 1.0 .. Float'Last =>

Put_Line ("Positive");

when Float'First .. -1.0 =>

Put_Line ("Negative");

when others =>

Put_Line ("Other");

end case;
```

Is this correct? (6/10)



```
for I in 0 .. 10 loop
    I := 10;
end loop;
```

Is this correct? (6/10)



Compilation Error, I is constant in the loop, assignment is forbidden

8

What is the output of this code? (7/10)

```
for I in 10 .. 0 loop
    Put_Line (Integer'Image (I));
end loop;
```

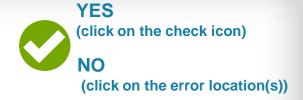
8

What is the output of this code? (7/10)

```
for I in 10 .. 0 loop
    Put_Line (Integer'Image (I));
end loop;
```

Nothing, the loop has an empty range, needs "reverse 0 .. 10" to be backwards.

Is this correct? (8/10)



```
if A != 0 then
   Put_Line ("A is not 0");
end if;
```

Is this correct? (8/10)



Compilation Error, the Ada inequality symbol is "/="

```
if A != 0 then
   Put_Line ("A is not 0");
end if;
```

Is this correct? (9/10)



```
I : Natural;
begin
  for I in 0 .. 10 loop
    null;
end loop;
```

Is this correct? (9/10)



```
I : Natural;
begin
  for I in 0 .. 10 loop
    null;
end loop;
```

Correct, although the I declaration is redundant

8

What is the output of this code? (10/10)

```
X : Integer := 1;
begin
for I in 1 .. X loop
    X := 10;
    Put_Line ("A");
end loop;
```

8

What is the output of this code? (10/10)

```
X : Integer := 1;
begin
for I in 1 .. X loop
    X := 10;
    Put_Line ("A");
end loop;
```

"A" / the range is evaluated before entering the loop





university.adacore.com