Numeric Pitfall in C/C++/Java

What is the Program Output ?

- This program compiles fine
- What is its output?

```
#include <stdio.h>

int main () {
  int length  = 8265;
  int width = 0252;
  int height  = 8292;

  printf ("length     = %d\n", length);
  printf ("width      = %d\n", width);
  printf ("height     = %d\n", height);
}
```


Surprised ?

```
C:\tmp>type main2.c
int main () {
  int length = 8265;
  int width  = 0252;
  int height = 8292;

  printf ("length = %d\n", length);
  printf ("width  = %d\n", width);
  printf ("height = %d\n", height);
}

C:\tmp>gcc main2.c

C:\tmp>a.exe
length = 8265
width  = 170
height = 8292

C:\tmp>
```

intent ?

Numbers in C/C++/Java

- In C/C++/Java numbers starting with 0 are octal numbers
- A bad choice
  - Error-prone
  - Hard-to-read
- There is no way to specify numbers in base 2
  - Very surprising giving the fact that C was meant for to be a lowlevel systems language
- Never use octal numbers

The Ada Version

```
with Text_IO;

procedure Main is
   Length : Integer := 8265;
   Width  : Integer := 0252;
   Height : Integer := 8292;
begin
   Text_IO.Put_Line (Length'img);
   Text_IO.Put_Line (Width'img);
   Text_IO.Put_Line (Height'img);
end Main;
```

```
C:\tmp>type main.adb
with Text_IO;

procedure Main is
   Length : Integer := 8265;
   Width  : Integer := 0252;
   Height : Integer := 8292;
begin
   Text_IO.Put_Line (Length'Img);
   Text_IO.Put_Line (Width'Img);
   Text_IO.Put_Line (Height'Img);
end Main;

C:\tmp>gnatmake main.adb
gcc -c main.adb
gnatbind -x main.ali
gnatlink main.ali

C:\tmp>main
 8265
 252
 8292

C:\tmp>
```

No surprises in Ada

Numbers in Ada

```
Length   : Integer := 8265;
Width    : Integer := 0252;     -- regular decimal number
Width_8  : Integer := 8#252#;   -- octal number

B_Mask   : Integer := 2#1100_1011#;   -- binary number
                                      -- you can use '_' to separate digits
W_Mask   : Integer := 16#FFF1_A4B0#; -- Hexadecimal number
```

- If no base is specified the number a decimal number
- In Ada you can specify any base from 2 to 16 (for both integer and real (floating point) numbers
- Use the "_" to separate digits for clarity
  - 1_000_000_000

Is the Following Code Correct ?

```
#include <limits.h>

/*  If *y is zero and x > 0 set *k to the biggest positive integer.
 *  If *y is zero and x <=0 leave *k  unchanged.
 *  If *y is not zero set *k to x/*y and increment *y by 1.
 */
void check_divide (int *k, int x, int *y)  {
   if (*y = 0)
     if (x > 0)
       *k = INT_MAX;
   else
     *k = x / *y; // it is safe to divide by *y since it cannot be 0
     *y++;
}
```

- This program compiles fine, but has a number of problems. Whichones?

There are 4 Bugs

- **= versus ==**
  - Using "=" for assignment and "==" for equality is a poor choice
  - Use a compiler that warns you when you use "=" inside tests
  - This is a hard problem because C++ style encourages the use of "=" inside tests:

```
while (*s1++ = *s2++);
```

- **Dangling `else` problem**
  - Always bracket everything
- **Nested y++**
- **Bad operator precedence: *y++ means *(y++)**
  - When in doubt use parentheses or separate tokens with white spaces ...

What about Java?

- **= versus ==**
  - This problem exists in Java for **Boolean**, it has been fixed for other data types
- **Dangling `else` problem**
  - Problem is still there
- **Bad operator precedence: *j++ means *(j++)**
  - No * operator in Java.
    This problem has been solved

```
boolean safety_flag;
boolean danger_flag;
...
if (safety_flag = danger_flag) {
    alarm ();
}
```

This is **OK** in **Java** but is often a bug

The Correct Version

```
#include <limits.h>

/*  If *y is null and x > 0 set *k to the biggest positive integer.
 *  If *y is null and x <=0 leave *k unchanged.
 *  If *y is non null set *k to x/*y and increment *y by 1.
 */
void check_divide (int *k, int x, int *y)  {
   if (*y == 0)  {
     if ( x > 0)
       *k = INT_MAX;
   }
   else  {
     *k = x / *y; // it is safe to divide by *y since it cannot be 0
     (*y)++;      // or  *y ++
   }
}
```


Ada Solves these Pitfalls

```
--  If Y = 0 and  X > 0  set K to the biggest positive integer.
--  If Y = 0 and  X <= 0  leave K unchanged.
--  If Y /= 0 set  K  to be  X  divided by Y and increment  Y by 1.

procedure Check_Divide  (K : in out Integer;  X : Integer;  Y : in out Integer) is
begin
   if  Y = 0  then
      if  X > 0  then
         K := Integer'Last;  -- K is set to the largest Integer
      end if;
   else
      K := X / Y;  -- it is safe to divide by Y since it cannot be 0
      Y := Y + 1;
   end if;
end Check_Divide;
```

Even Simpler: "elsif" "and then"

```
-- If Y = 0 and X > 0 set K to the biggest positive integer.
-- If Y = 0 and X <= 0 leave K unchanged.
-- If Y /= 0 set K to be X divided by Y and increment Y by 1.

procedure Check_Divide (K : in out Integer; X : Integer; Y : in out Integer) is
begin
   if Y = 0 and then X > 0 then
      K := Integer'Last;  -- K is set to the largest Integer
   elsif Y /= 0 then
      K := X / Y;  -- it is safe to divide by Y since it cannot be 0
      Y := Y + 1;
   end if;
end Check_Divide;
```


Lexical & Syntactic Clarity

- = in Ada means equality while := means assignment
  - If you use one instead of the other you get a compiler error
- No dangling else in Ada
  - ```
    if ... then
       ...  -- Must be terminated by an end if;
    end if;
    ```
  - If it isn't then you get a compiler error
- In Ada comments start with -- and go to the end of the line. No other type of comment
- No ++ operators in Ada
- No need for a * operator in Ada

Ada Core TECHNOLOGIES.INC

ACT EUROPE

Operator Precedence & Association Pitfalls in C/C++/Java

Operator Precedence & Association

CAUTION

`if (x & mask == 0)` means `if (x & (mask == 0) )`

- It does NOT mean `if ((x & mask) == 0)`

CAUTION

`if (x < y < z)` means `if ((x < y) && (1 < z)) || (0 < z))`

- It does NOT mean `if ((x < y) && (y < z))`

- This is not an exhaustive list. Examples
  - What does `hi << 4 + low` mean?
  - What does `x == y == z` mean?

Ada Solution

- `if x and mask = 0 ...` means `if (x and mask) = 0 ...`
- `if x < y < z ...` — Gives you a compiler error in Ada
  - You have to write `if X < Y and Y < Z ...`

Ada Core TECHNOLOGIES.INC

ACT EUROPE

More C/C++/Java Syntactic Pitfalls
    ```

Is this Code Correct ?

```
//   If the signal ahead is clear then increase the speed.

void increase_speed_if_safe (int speed, int signal)  {
  if (signal == CLEAR);
    increase_speed ();
}
```

- This program compiles fine, but has a problem. Which one?

Beware of Spurious Semicolons

```
//   If the signal ahead is clear then increase the speed.

void increase_speed_if_safe (int speed, int signal)  {
  if (signal == CLEAR);
    increase_speed ();
}
```


The Ada Version is Always Safe

```
-- If the signal ahead is clear then increase the speed.

procedure increase_speed_if_safe (speed : integer; signal : integer) is
begin
   if signal = CLEAR then
      increase_speed;
   end if;
end increase_speed_if_safe;
```

- If you write
  if signal = CLEAR then ;
  - You get a compiler error

Enumerations and Switch

```
enum  Alert_Type {LOW, MEDIUM, HIGH, VERY_HIGH};
// This is C or C++.
// Java does not have enumerations, you have to use int instead

void handle_alert (enum  Alert_Type  alert) {
   switch (alert) {
     case LOW:
       activate_camera ();
     case MEDIUM:
       send_guard ();
     case HIGH:
       sound_alarm ();
   }
}

void process_alerts () {
   handle_alert (2);
   …
```

- This program compiles fine, but has a number of problems. Whichones?

Code Defects

- Don't forget break statements
- C/C++/Java do not check that you have treated all cases in the switch
- case labels can be integers or (values of) any enum type, not just enum Alert_Type which in most cases will be an error

```
void handle_alert (enum  Alert_Type
alert) {
   switch (alert) {
     case LOW:
       activate_camera ();
       break;
     case MEDIUM:
       send_guard ();
       break;
     case HIGH:
       sound_alarm ();
       break;
     case VERY_HIGH:
       alert_police ();
       break;
   }
}
void process_alerts () {
   handle_alert (HIGH);
```


Ada is Safer

```
type Alert_Type is (LOW, MEDIUM, HIGH, VERY_HIGH);

procedure Process_Alert (Alert : Alert_Type) is
begin
   case Alert is
      when LOW       =>
         Activate_Camera;
      when MEDIUM    =>
         Send_Guard;
      when HIGH      =>
         Sound_Alarm;
      when VERY_HIGH =>
         Alert_Police;
   end case;
end Process_Alert;
```

- No break statements
- Ada will check that you have treated all cases in the case statement
- You can only use an object of type Alert_Type

Enumeration Types in Ada

```
type Alert is (LOW, MEDIUM, HIGH, VERY_HIGH);

procedure P (B : Integer) is
   A : Alert;
begin
   A := B;
```

Compilation error

- Enumerations are true types in Ada
- In C enums are just integers
- In C++ enums are implicitly converted to ints (not from ints)

```
// C++
enum Alert {LOW, MEDIUM, HIGH, VERY_HIGH};

int   k = LOW;      // accepted by C++
Alert a = 1;        // rejected by C++
```


No Enumerations in Java :(

- In Java you must use integers

```
// Java
public static final int LOW      = 0;
public static final int MEDIUM   = 0;
public static final int HIGH     = 0;
public static final int VERY_HIGH = 0;
```


Enumeration Types and Attributes in Ada

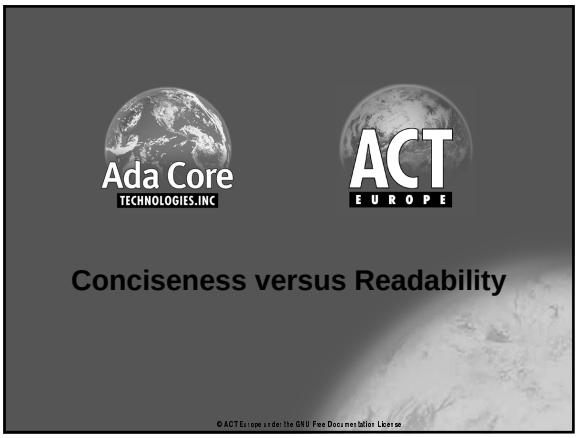
Predefined attributes

Alert_Type'First	LOW
Alert_Type'Last	VERY_HIGH

Predefined enumerations

```
type Boolean   is (False, True);

type Character is (..., 'a', 'b', 'c', ...);
```

Always Favor Readability

- What does the following mean:
  - This is valid C/C++/Java
  - It is very concise ...
  - ... but not very readable

```
int  x;
int  y;
int  z;
…
x = y---z--;
```

- What does the following mean:
  - This is valid Ada
  - It is less concise ...
  - ... but very readable

```
x : integer;
y : integer;
z : integer;
…
x := y - z;
y := y - 1;
z := z - 1;
```


Good Programming is Altruistic

- Other people read and work on your code
- Write your code so that others can read and work on it
- Readability is a function of the programmers in your group
  - Present AND future programmers
- If you are in a group of gurus it is ok to prefer `x = y-- - z--;`
- In most organizations programmers are not gurus
- In this case

```
x = y - z;
y--;
z--;
```

is easier to read/work on

Conciseness and Non Determinism

- What does the following mean:
  - It compiles, but ...
  - ... Its semantics are undefined

```
{
    int k = 0;
    int v [10];

    k = v [k++];
}
```

- It could mean any one of the following (when written in Ada):

```
declare
    K : Integer := 0;
    V : array (0 .. 9) of Integer;
begin
    K := V (K);
    K := K + 1;
    ...
```

```
declare
    K : Integer := 0;
    V : array (0 .. 9) of Integer;
begin
    K := V (K);
    ...
```

```
declare
    K : Integer := 0;
    V : array (0 .. 9) of Integer;
begin
    Erase_All_Hard_Disks;
    ...
```


If Nobody Can Read It Don't Write It

- A piece of code is written once ....
- ... but read and modified many many times
- C/C++/Java syntax favors conciseness over readability
  - This leads to bugs and wasted time in debugging
  - This means that software is more costly to develop or is buggy or both
- Ada syntax favors readability
  - Ada compiler catches silly mistakes
  - Faster and cheaper to produce correct code when written in Ada