Trabajo Práctico N° 07

Compatibilidades de tipos entre variables

Pasaje de Parámetros en ADA

- La forma de pasar parámetros en **ADA** se realiza anteponiendo en la definición de los parámetros (parámetros formales) algunos de los siguientes prefijos: **IN**, **OUT** o **IN OUT**. Si se omite alguno de ellos, se toma por defecto que es **IN**. La definición de cada forma de pasar los parámetros es la siguiente:

IN	El parámetro formal es una constante y permite sólo lectura del valor asociado al parámetro actual.
IN OUT	El parámetro formal es una variable y permite tanto leer como modificar el valor asociado al parámetro actual.
OUT	El parámetro formal es una variable y permite sólo modificar el valor asociado al parámetro actual.

Conversiones explícitas en asignaciones entre variables en ADA:

- El lenguaje **ADA** tiene **compatibilidad por nombre**: los elementos son compatibles si los nombres de los tipos son iguales.
- En la mayoría de los lenguajes las conversiones son implícitas. **ADA** requiere que las conversiones sean explícitas, de lo contrario arrojan errores de compilación.
- Las conversiones explícitas se permiten entre tipos relacionados como los definidos a continuación:

```
conversión_tipo ::= tipo_indicado (expresión_a_convertir)
```

Aclaración General para la Resolución del Práctico:

- Los enlaces son solo una ayuda para facilitar la ejecución de los códigos a los estudiantes.
- Si el intérprete o compilador del enlace no está disponible, es posible utilizar cualquier intérprete o compilador instalado de forma local con el código de prueba o los listados en la sección Material de la página de la cátedra. Por temas relacionados a Material escribir a José M. Massa
- El sitio de Typescript puede direccionar, en algún intento, a sitios propios, lo cual se soluciona en otros intentos
- En el caso de Javascript, dependiendo si se ejecuta con salida en consola o en un html es posible utilizar document.writeln(...); o console.log(...) según corresponda;
- El programa en Dart y Ruby deberá ejecutarlo solamente con el intérprete y máquina virtual oficial ya que en muchos casos las otras versiones online no tienen el intérprete oficial

Ejercicio 1

Considerando el siguiente código en un lenguaje con alcance estático que solamente tiene los tipos int y float.

```
a = 1
b = 2.1
c = 4
c = a + b
print c
```

De acuerdo a cada tipo de salida, completar la siguiente tabla especificando para cada tipo de lenguaje:

a) Si la salida es posible o no
b) El momento (Compilación/Ejecución) durante el cual existen las tablas de símbolos y de compatibilidad de tipos para la operación de suma
c) Si el lenguaje realiza conversiones del lado derecho, en ese caso. Especificar cuáles y en qué momento

#	Salida	Estático	Tipo Algol	Dinámico
s1	3.1			
s2	3			
s3	Error: No es posible sumar int y float			
s4	Error: No es posible asignar un float a un int			

Ejercicio 2

Considerando el resultado de la ejecución de cada uno de los siguientes códigos en diferentes lenguajes de programación (el título con el nombre del lenguaje es un enlace a un código disponible online):

Python 3	Perl	PHP	Javascript

a = 1
b = "Hola"
c = 2.1
print(a,type(a))
a = b
print(a,type(a))
a = c
print(a,type(a))
	$a = 1;
	$b = "Hola";
	$c = 2.1;
	print($a, "\n");
	$a = $b;
	print($a, "\n");
	$a = $c;
	print($a, "\n");
		<?php
		$a = 1;
		$b = "Hola";
		$c = 2.1;
		echo $a,gettype($a), "\n";
		$a = $b;
		echo $a,gettype($a), "\n";
		$a = $c;
		echo $a,gettype($a), "\n";
		?>
			a = 1;
			b = "Hola";
			c = 2.1;
			document.writeln(a,typeof(a));
			a = b;
			document.writeln(a,typeof(a));
			a = c;
			document.writeln(a,typeof(a));

Dart	**Typescript**	**Ruby**	**Scheme**

void main() {
  var a = 1;
  var b = "Hola";
  var c = 2.1;
  var d;
  int e;
	var a = 1;
	var b = "Hola";
	var c = 2.1;
	var d;
	var e : number;
	document.writeln(typeof(a));
		a = 1
		b = "Hola"
		c = 2.1
		p a,a.class
		a = b
		p a,a.class
			(define a 1)
			(define b "Hola")
			(define c 2.1)
			(display a)
			(display (class-of a))
			(define a b)

```
  d = 1;
  print(a.runtimeType);
  a = b;
  print(a.runtimeType);
  a = c;
  print(a.runtimeType);
  print(d.runtimeType);
  d = 3.1;
  print(d.runtimeType);
  e = 1.1;
  print(d.runtimeType);
}
```

```
a = b;
document.writeln(typeof(a));
a = c;
document.writeln(typeof(a));
d = a;
document.writeln(typeof(d));
d = b;
document.writeln(typeof(d));
d = c;
document.writeln(typeof(d));
e = b;
```

```
a = c
p a,a.class
```

```
(display a)
(display (class-of a))
(define a c)
(display a)
(display (class-of a))
```

Java

```
public class
HelloWorld{
    public static void
main(String []args){
      int a = 1;
      String b = "Hola";
      double c = 2.1;

System.out.println(a);
      a = b;

System.out.println(a);
      a = c;

System.out.println(a);
    }
}
```

C++

```
#include <iostream>
using namespace std;
int main()
{
 int a = 1;
 char b[256] = "Hola";
 double c = 2.1;
 cout << typeid(a).name() <<
endl;
 a = b;
 cout << typeid(a).name() <<
endl;
 a = c;
 cout << typeid(a).name() <<
endl;
auto d = 1;
cout << typeid(d).name() <<
endl;
d = b;
cout << typeid(d).name() <<
endl;
return 0;
}
```

C#

```
using System.IO;
using System;
class Program
{
  static void Main()
  {
    int a = 1;
    String b = "Hola";
    double c = 2.1;
Console.WriteLine(a.GetType().Name)
;
a = b;
Console.WriteLine(a.GetType().Name)
;
a = c;
Console.WriteLine(a.GetType().Name)
;
var d = a;
d = b;
   }
}
```

Rust

```
fn main() {
  let mut a : i32 = 1;
  let mut b : String =
"Hola".to_string();
  let mut c = 2.1;
  println!("{}",a);
  a = b;
  println!("{}",a);
  a = c;
  println!("{}",a);
}
```

Go

```
package main
import "fmt"
func main() {
  var a int32 = 1
  var b string = "Hola"
  var c float32 = 2.1
  fmt.Printf("%T\n", a)
  a = "Que tal"
  fmt.Printf("%T\n", a)
  a = 3.1
  fmt.Printf("%T\n", a)
  a = b
  fmt.Printf("%T\n", a)
  a = c
  fmt.Printf("%T\n", a)
  a = 1
  fmt.Printf("%T\n", a)
```

Swift

```
import Foundation
var a : Int = 1
var b = "Hola"
var c = 2.1
print(type (of: a))
a = b
print(type (of: a))
a = c
print(type (of: a))
```

Ada

```
with Ada.Text_IO; use Ada.Text_IO;
procedure abc is
a : Integer;
b : Float;
c : String(1 .. 4);
begin
  a := 1;
  b := 2.1;
  c := "Hola";
  a := b;
  a := c;
end abc;
```

R

```
a <- 1
b <- "Hola"
c <- 2.1
typeof(a)
a <- b
typeof(a)
a <- c
typeof(a)
```

d := a;
d = b;
}			

Importante:

- Para resolver el ejercicio **leer las Aclaraciones Generales** de la página 1 del práctico
- Es posible que para responder los incisos que se encuentran debajo, se deban comentar las líneas que dan error

Responder, para cada caso, los siguientes incisos de acuerdo a lo que puede deducir del ejemplo, fundamentando brevemente las respuestas:

a) ¿Qué clase de binding de tipos tiene el lenguaje? (estático/dinámico)
b) Si el lenguaje tiene o no inferencia de tipos en asignaciones y si la inferencia se produce en tiempo de compilación o ejecución.
c) En los casos de binding de tipo estático, identificar aquellos donde sea posible asignar un valor de tipo punto flotante a una variable de tipo entero e indicar si la conversión es implícita o explícita.
d) ¿El lenguaje tiene tipos diferentes para los valores entero y punto flotante? (si/no/no se sabe)
e) ¿El lenguaje tiene palabras clave o anotaciones de nombres de tipos?
f) ¿El lenguaje tiene algún tipo de verificación de compatibilidad entre tipos para la asignación?

Ejercicio 3

Considerando los siguientes códigos en diferentes lenguajes de programación:

Python 3	Perl	PHP	Javascript

a = 1
b = "Hola"
c = 2.1
d = a + b
print(type(d))
 	
e = a + c
print(type(e))
f = b + c
print(type(f))
	$a = 1;
	$b = "Hola";
	$c = 2.1;
	$d = $a + $b;
	print($d,"\n");
	$e = $a + $c;
	print($e,"\n");
	$f = $b + $c;
	print($f,"\n");
		$a = 1;
		$b = "0Hola";
		$c = 2.1;
		$d = $a + $b;
		echo $d, " ",
		gettype($d), "\n";
		$e = $a + $c;
		echo $e, " ",
		gettype($e), "\n";
		$f = $b + $c;
		echo $f, " ",
		gettype($f), "\n";
			a = 1;
			b = "Hola";
			c = 2.1;
			d = a + b;
			document.writeln(d);
			document.writeln(typeof(d));
			e = a + c;
			document.writeln(e);
			document.writeln(typeof(e));
			f = b + c;
			document.writeln(f);
			document.writeln(typeof(f));

Dart	**Typescript**	**Ruby**	**R**

void main() {
var a = 1;
var b = "Hola";
var c = 2.1;
var d = a + b;
print(d.runtimeType);
var e = a + c;
print(e.runtimeType);
var f = b + c;
print(f.runtimeType);
}
	var a = 1;
	var b = "Hola";
	var c = 2.1;
	var d = a + b;
	document.writeln(typeof(a));
	var e = a + c;
	document.writeln(typeof(e));
	var f = b + c;
	document.writeln(typeof(f));
		a = 1
		b = "Hola"
		c = 2.1
		d = a + b
		p d.class
		e = a + c
		p e.class
		f = b + c
		p f.class
			a <- 1
			b <- "Hola"
			c <- 2.1
			d <- a + b
			typeof(d)
			e <- a + c
			typeof(e)
			f <- b + c
			typeof(f)

Para cada caso, responder los siguientes incisos con el tipo de compatibilidad para la suma que posee el lenguaje (SI/NO) y el tipo en los términos en los cuales se hace la operación, fundamentando brevemente las respuestas:

a) Suma entre enteros y string
b) Suma entre enteros y punto flotante
c) Suma entre punto flotante y string

Ejercicio 4

Dado el siguiente programa en lenguaje ADA 2012 o superior (es posible ejecutarlo escribiendo el programa en el cuadro de código del enlace oficial de ADA https://learn.adacore.com/index.html):

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Ejemplo_Contrato is
  function Doble (A : Integer) return Integer is
    (A * A)
    with
        Pre  => (A < 1000),
        Post => (Doble'Result < 100000);
  V : Integer;
begin
  V := Doble (10);
  Put_Line ("El doble de 10 es " & Integer'Image (V));
  V := Doble (1001);
  Put_Line ("El doble of 1001 es " & Integer'Image (V));
  V := Doble (900);
  Put_Line ("El doble of 900 es " & Integer'Image (V));
end Ejemplo_Contrato;
```

a) Definir en cuáles líneas se generan errores debido a que no se cumplen las precondiciones, si esos errores se producen en tiempo de compilación o ejecución y explicar por qué se producen esos errores.

b) Definir en cuáles líneas se generan errores debido a que no se cumplen las postcondiciones, si esos errores se producen en tiempo de compilación o ejecución y explicar por qué se producen esos errores.