Ada/SPARK, el lenguaje cuando las cosas tienen que funcionar

Una breve introducción de Ada/SPARK

Fernando Oleo Blanco - Irvise

Disclaimer

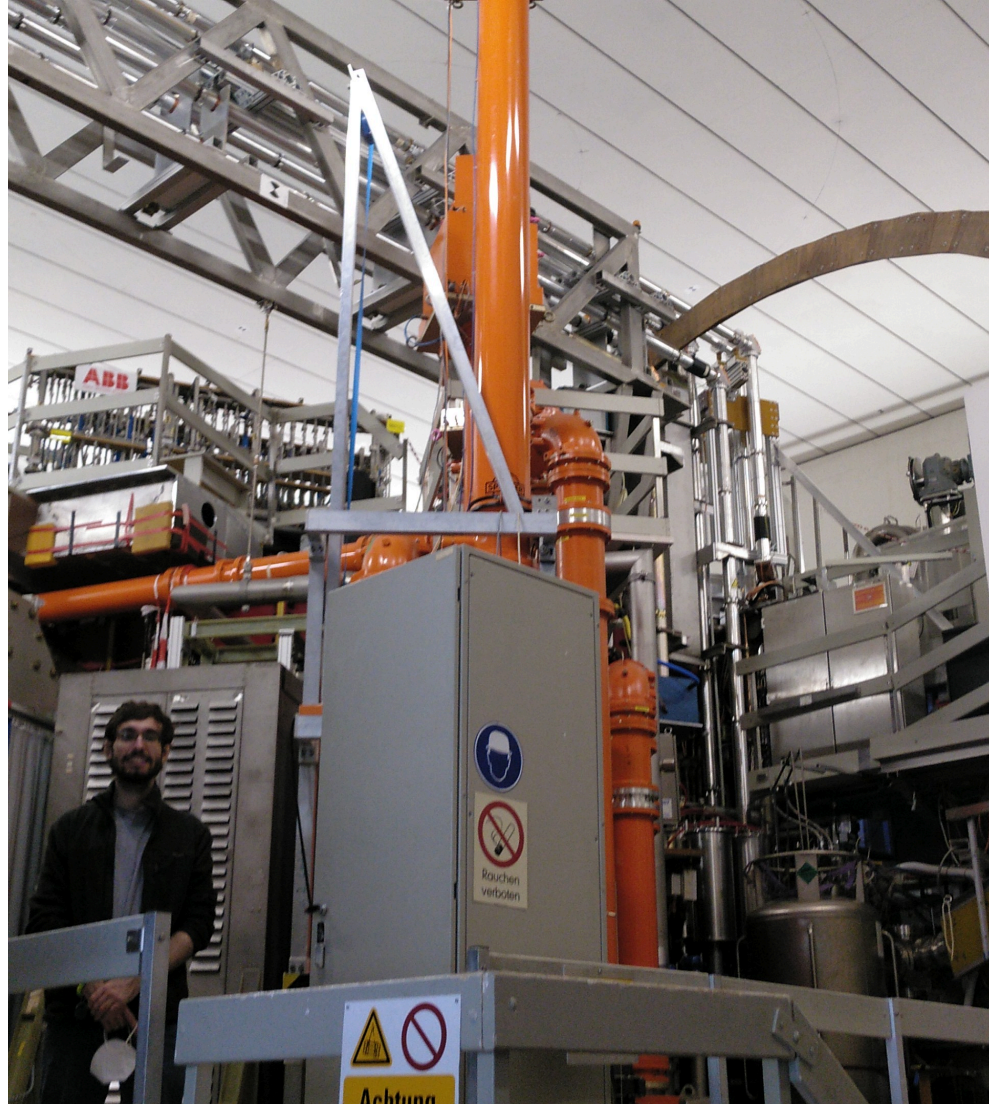
No va a ser muy seria la charla
¡También tiene que ser entretenida!

Mis opiniones son solo mías

Tabla de contenidos

whoami

- Fer, el original desde 1997 :)
  - Irvise en la red
- Ingeniero industrial y mecánico
  - Materiales, sistemas, FEM...
- Actualmente trabajando en el sector nuclear
  - Ingeniero de sistemas (P&IDs, ciclos, componentes...), ¡nada de programación!
- Enorme fan del Libre Software
  - Linux full-time desde los 18
- ¿Y Ada dónde aparece?



Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

¿Podemos hacerlo mejor?

Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

¿Podemos hacerlo mejor?

- **Fortran 2003-2023**
- C/C++? Rust?
- Frama-C, seL4, Isabelle/HOL, Rocq, F*?

Real Programmers Don't Use PASCAL

Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

¿Podemos hacerlo mejor?

- **Fortran 2003-2023**
- C/C++? Rust?
- Frama-C, seL4, Isabelle/HOL, Rocq, F*?

Real Programmers Don't Use PASCAL

Y entonces Ada...

- "Formalmente verificable con SPARK"
- "Legible y fácil de aprender"
- "Eficiente, de alto y bajo nivel, compilado, sin GC"
- "Utilizado en aeroespaciales, militar, sistemas críticos..."
- ...

Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

¿Podemos hacerlo mejor?

- **Fortran 2003-2023**
- C/C++? Rust?
- Frama-C, seL4, Isabelle/HOL, Rocq, F*?

Real Programmers Don't Use PASCAL

Y entonces Ada...

- "Formalmente verificable con SPARK"
- "Legible y fácil de aprender"
- "Eficiente, de alto y bajo nivel, compilado, sin GC"
- "Utilizado en aeroespaciales, militar, sistemas críticos..."
- ...

Mmmm... 🤔🧐

```
with Ada.Text_IO;

procedure My_Hello_World is
begin
   Ada.Text_IO.Put_Line ("Hello, World!");
end My_Hello_World;
```

Fortran 66

FRAPTRAN-2.0, analisis de seguridad nuclear

Muy bien escrito, documentado y estructurado...

```
      COMMON A/B
C     COMMENT
      X=KODE(J)
      IF (X.LT.0) GO TO 55
      IF (X.EQ.0) GO TO 55
      GO TO 40

55    CALL SOMETHING(INTPUT, OUTFLOAT)
```

¿Podemos hacerlo mejor?

- **Fortran 2003-2023**
- C/C++? Rust?
- Frama-C, seL4, Isabelle/HOL, Rocq, F*?

Real Programmers Don't Use PASCAL

```
type UART_Type is record
   txdata : txdata_Type;
   rxdata : rxdata_Type;
   txctrl : txctrl_Type;
   rxctrl : rxctrl_Type;
end record
   with Size => 3 * 32,
        Full_Volatile_Access;
for UART_Type use record
   txdata at 16#00# range 0  .. 31;
   rxdata at 16#04# range 0  .. 31;
   txctrl at 16#08# range 0  .. 15;
   rxctrl at 16#08# range 16 .. 31;
end record;
UART0_ADDRESS : constant := 16#1001_3000#;
UART0 : aliased UART_Type
   with Address  => System'To_Address
                    (UART0_ADDRESS),
        Volatile => True,
        Import   => True;
```

```
type UART_Type is record
   txdata : txdata_Type;
   rxdata : rxdata_Type;
   txctrl : txctrl_Type;
   rxctrl : rxctrl_Type;
end record
   with Size => 3 * 32,
        Full_Volatile_Access;
for UART_Type use record
   txdata at 16#00# range 0  .. 31;
   rxdata at 16#04# range 0  .. 31;
   txctrl at 16#08# range 0  .. 15;
   rxctrl at 16#08# range 16 .. 31;
end record;
UART0_ADDRESS : constant := 16#1001_3000#;
UART0 : aliased UART_Type
   with Address  => System'To_Address
                    (UART0_ADDRESS),
        Volatile => True,
        Import   => True;
```

¿Por qué una presentación sobre Ada?

¿Por qué una presentación sobre Ada?

- La gente no sabe lo potente que es Ada/SPARK
  - Hoy solo veremos lo básico basiquísimo... Pero las partes avanzadas de Ada es lo que le hace brillar
  - Es muy elegante, se centra en solucionar los problemas

¿Por qué una presentación sobre Ada?

- La gente no sabe lo potente que es Ada/SPARK
  - Hoy solo veremos lo básico basiquísimo... Pero las partes avanzadas de Ada es lo que le hace brillar
  - Es muy elegante, se centra en solucionar los problemas
- Haré comparaciones con C/C++ y Rust para que se entienda mejor

¿Por qué una presentación sobre Ada?

- La gente no sabe lo potente que es Ada/SPARK
  - Hoy solo veremos lo básico basiquísimo... Pero las partes avanzadas de Ada es lo que le hace brillar
  - Es muy elegante, se centra en solucionar los problemas
- Haré comparaciones con C/C++ y Rust para que se entienda mejor
- Porque Ada/SPARK es preciosa

¿Por qué una presentación sobre Ada?

- La gente no sabe lo potente que es Ada/SPARK
  - Hoy solo veremos lo básico basiquísimo... Pero las partes avanzadas de Ada es lo que le hace brillar
  - Es muy elegante, se centra en solucionar los problemas
- Haré comparaciones con C/C++ y Rust para que se entienda mejor
- Porque Ada/SPARK es preciosa

¿NVIDIA?


NVIDIA — Securing the Future of Safety and Security of Embedded Software (
AdaCore
Watch on

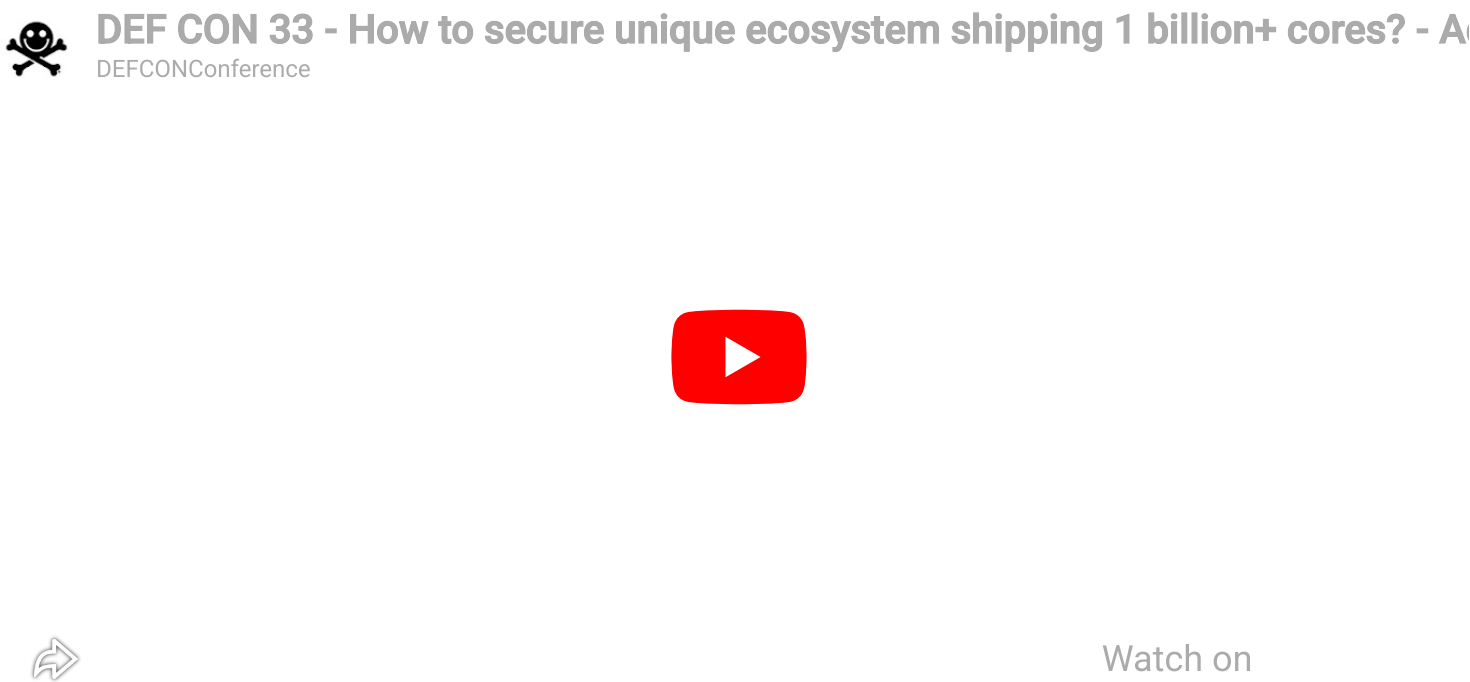
¿NVIDIA?

DEF CON 33 - How to secure unique ecosystem shipping 1 billion+ cores? - A

DEFCONConference

Watch on

¿NVIDIA?



NVIDIA ISO-26262 SPARK guidelines. Además, NVIDIA certifica a SIL-4, 7 MLOC OS escrito en Ada/SPARK (2025). Más información en this ACM article (2024).

La historia de Ada

Nacimiento, caída y renacimiento

Departamento de la defensa en los 70, USA

Grandes problemas con el software

- Cada equipo usaba herramientas distintas
- Bugs por todos los lados
- Altos costes de mantenimiento
- Dificultad de integrar nueva gente

Departamento de la defensa en los 70, USA

Grandes problemas con el software

- Cada equipo usaba herramientas distintas
- Bugs por todos los lados
- Altos costes de mantenimiento
- Dificultad de integrar nueva gente

Solución, un entorno de programación integrado

Strawman, woodman... y Ada

Departamento de la defensa en los 70, USA

Grandes problemas con el software

- Cada equipo usaba herramientas distintas
- Bugs por todos los lados
- Altos costes de mantenimiento
- Dificultad de integrar nueva gente

Solución, un entorno de programación integrado

Strawman, woodman... y Ada

En resumen... la lista de la compra

- Que sea fácil de aprender
- Que sea fácil de comprobar
- Que se detecten fallos cuanto antes
- Que sea de alto rendimiento
- Que corra en sistemas empotrados
- Que sea multiproceso
- Que sea claro y demuestre la especificación
- Que sea de estándar abierto

Departamento de la defensa en los 70, USA

Grandes problemas con el software

- Cada equipo usaba herramientas distintas
- Bugs por todos los lados
- Altos costes de mantenimiento
- Dificultad de integrar nueva gente

Solución, un entorno de programación integrado

Strawman, woodman... y Ada

En resumen... la lista de la compra

- Que sea fácil de aprender
- Que sea fácil de comprobar
- Que se detecten fallos cuanto antes
- Que sea de alto rendimiento
- Que corra en sistemas empotrados
- Que sea multiproceso
- Que sea claro y demuestre la especificación
- Que sea de estándar abierto

No todo fueron rosas

Costes altos, lenguaje limitado, bugs en compiladores, muy distinto a otros lenguajes...

Ada en comparación

Ada ex un estandar ISO abierto (ISO/IEC 8652)

- Ada 83 (ANSI/MIL-STD-1815A): 348 páginas (US letter)
- Ada 2012: 832 páginas (A4)
- Ada 2022: 1048 páginas (A4)
- Hay una versión anotada ("Rationale") (AARM) que explica las decisiones y el diseño
- No necesita de *MISRA* para sistemas críticos (C++:23, 253 páginas)
- **¡¡Diseñado para que sea leído por todos los programadores de Ada!!**

Ada en comparación

Ada ex un estandar ISO abierto (ISO/IEC 8652)

- Ada 83 (ANSI/MIL-STD-1815A): 348 páginas (US letter)
- Ada 2012: 832 páginas (A4)
- Ada 2022: 1048 páginas (A4)
- Hay una versión anotada ("Rationale") (AARM) que explica las decisiones y el diseño
- No necesita de *MISRA* para sistemas críticos (C++:23, 253 páginas)
- **¡¡Diseñado para que sea leído por todos los programadores de Ada!!**

C/C++ son estándar ISO

- C90 (ISO/IEC 9899): 219 páginas (A4)
- C23: 758 páginas (A4)
- C++98 (ISO/IEC 14882): 732 páginas (A4)
- C++23: 2104 páginas (A4)
- El estándar no está diseñado para ser leído por los usuarios (ISOCPP). Borradores disponibles en Github

Ada en comparación

Ada ex un estandar ISO abierto (ISO/IEC 8652)

- Ada 83 (ANSI/MIL-STD-1815A): 348 páginas (US letter)
- Ada 2012: 832 páginas (A4)
- Ada 2022: 1048 páginas (A4)
- Hay una versión anotada ("Rationale") (AARM) que explica las decisiones y el diseño
- No necesita de *MISRA* para sistemas críticos (C++:23, 253 páginas)
- **¡¡Diseñado para que sea leído por todos los programadores de Ada!!**

C/C++ son estándar ISO

- C90 (ISO/IEC 9899): 219 páginas (A4)
- C23: 758 páginas (A4)
- C++98 (ISO/IEC 14882): 732 páginas (A4)
- C++23: 2104 páginas (A4)
- El estándar no está diseñado para ser leído por los usuarios (ISOCPP). Borradores disponibles en Github

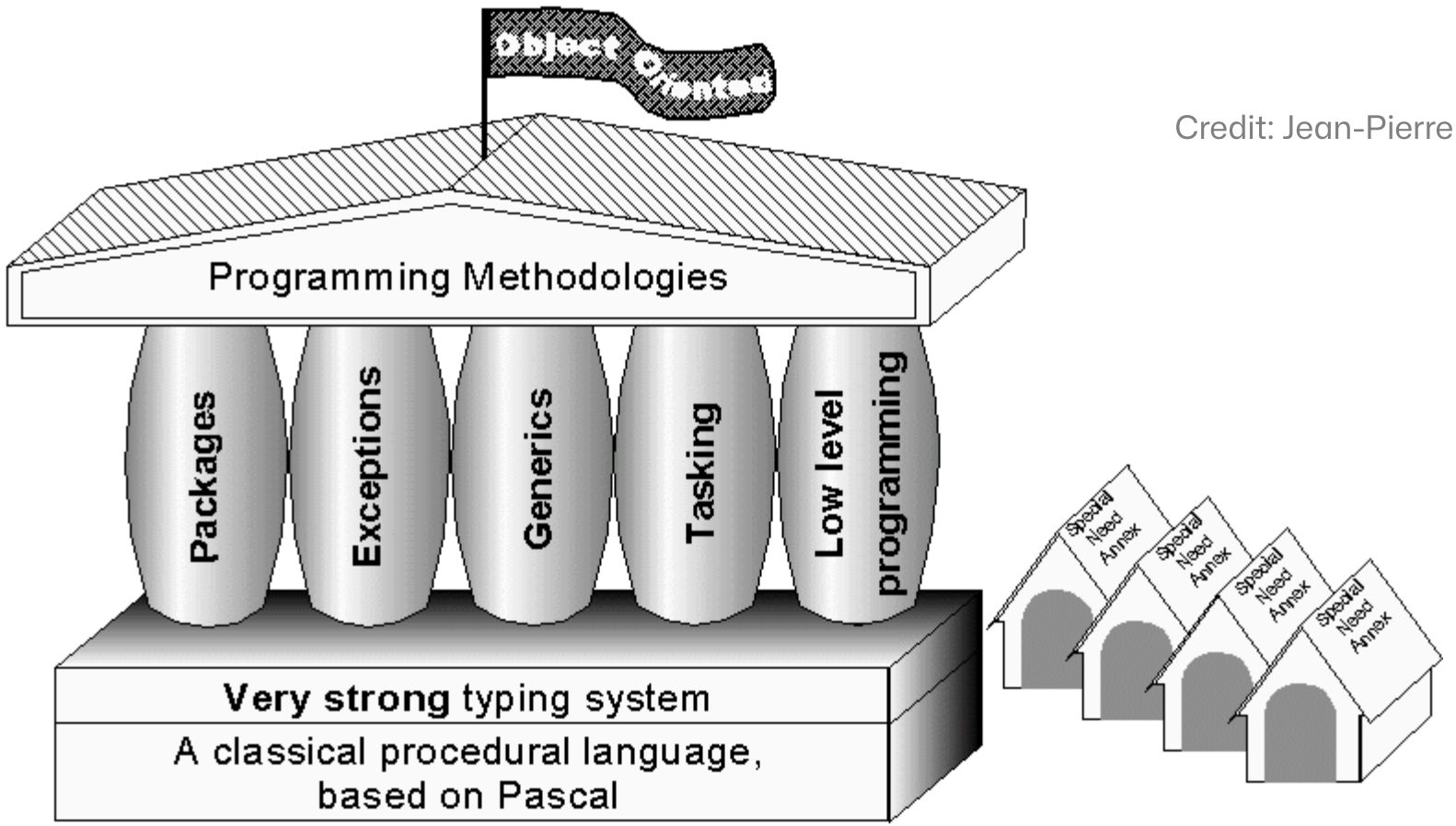
Rust *es* un estándar (no ISO)

- Creado en 2023. Ver The Rust Reference.

Ada/SPARK

Ya por fin, el contenido

Una imagen vale más que mil palabras



Credit: Jean-Pierre Rosen

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    float life;
} Player;
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    float life = 100.0; // C++11
} Player;
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Player_T is record
   ...
   Life : Float; -- Naïve approach
end record;  -- Ada 83

Player : Player_T;
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T;
end record;  -- Ada 83

Player : Player_T;
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := 100.0;
end record;  -- Ada 83

Player : Player_T;
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
Player.Life :=  1000.0; -- Runtime exception
                        -- Comp-time warning
                        -- SPARK: error
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
Player.Life :=  1000.0; -- Runtime exception
                        -- Comp-time warning
                        -- SPARK: error
Player.Life := -1000.0; -- Same as above
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
Player.Life :=  1000.0; -- Runtime exception
                        -- Comp-time warning
                        -- SPARK: error
Player.Life := -1000.0; -- Same as above
Player.Life :=  100;    -- Compile time error
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

Ada

```
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
Player.Life :=  1000.0; -- Runtime exception
                        -- Comp-time warning
                        -- SPARK: error
Player.Life := -1000.0; -- Same as above
Player.Life :=  Life_T(100);
```

¡Tipos, tipos, tipos y... más tipos! ¡Y datos!

Un ejemplo, ¿Cómo definiríais la vida de un jugador en un videojuego?

C++

```cpp
struct Player_t {
    ...
    const float  MAX_LIFE = 100.0;
    float life = MAX_LIFE; // C++11
} Player;
...

Player.life =  1000.0; // All good?
Player.life = -1000.0; // All good!?
Player.life =  100;    // Oh, yeah, that...
```

Implicit conversions considered harmful - Jason Turner - NDC TechTown 2025

Ada

```ada
type Life_T is digits 3 range 0.0 .. 100.0;
type Player_T is record
   ...
   Life : Life_T := Life_T'Last;
end record;  -- Ada 83

Player : Player_T;
Player.Life :=  1000.0; -- Runtime exception
                        -- Comp-time warning
                        -- SPARK: error
Player.Life := -1000.0; -- Same as above
--  Note: runtime checks can be turned off,
--  but it is not recommended!
```

El atlas de tipos: los básicos

Ada

```ada
-- Ada 83
type My_Enum  is (On, Off, Unknown);          -- Enum, it is not numeric!!!

type My_Int   is range -10 .. 2**8;           -- Integer, range must be increasing
type My_Float is digits 10;                   -- 100.00003, "digits" is the precission
type My_Fl2   is digits 10 range  0.0 .. 1.0; -- 0.3634

type My_Fix   is delta 0.5 range -1.5 .. 2.0; -- Fixed point numbers!!

-- Ada 95
type My_Mod     is mod 3;                        -- 0, 1, 2, 0, 1, 2...
type My_Decimal is delta 10.0 ** (-4) digits 20; -- Decimal types!
```

El atlas de tipos: los básicos

Ada

```ada
-- Ada 83
type My_Enum  is (On, Off, Unknown);          -- Enum, it is not numeric!!!

type My_Int   is range -10 .. 2**8;           -- Integer, range must be increasing
type My_Float is digits 10;                   -- 100.00003, "digits" is the precission
type My_Fl2   is digits 10 range  0.0 .. 1.0; -- 0.3634

type My_Fix   is delta 0.5 range -1.5 .. 2.0; -- Fixed point numbers!!

-- Ada 95
type My_Mod     is mod 3;                        -- 0, 1, 2, 0, 1, 2...
type My_Decimal is delta 10.0 ** (-4) digits 20; -- Decimal types!
```

Todas las declaraciones son distintas.

Esto es importante para los tipos genéricos

El atlas de tipos: las relaciones I

Utilizando relaciones (*sans objets*)

Ada

```ada
-- Ada 83
type    Grade   is range 0 .. 10;
subtype Failure is Grade range 0 .. 4; -- Compatible with Grade, but restricted range
```

El atlas de tipos: las relaciones I

Utilizando relaciones (*sans objets*)

Ada

```ada
-- Ada 83
type    Grade   is range 0 .. 10;
subtype Failure is Grade range 0 .. 4; -- Compatible with Grade, but restricted range

type Math_Grade is new Grade;          -- Binary ideantical to Grade, incompatible
type Econ_Grade is new Grade;          -- Binary ideantical to Grade, incompatible
```

El atlas de tipos: las relaciones I

Utilizando relaciones (*sans objets*)

Ada

```ada
-- Ada 83
type    Grade   is range 0 .. 10;
subtype Failure is Grade range 0 .. 4; -- Compatible with Grade, but restricted range

type Math_Grade is new Grade;          -- Binary ideantical to Grade, incompatible
type Econ_Grade is new Grade;          -- Binary ideantical to Grade, incompatible

type Sports_Grade is new Grade range 1 .. 6; -- Binary identical to Grade,
                                             -- incompatible with all others
                                             -- restricted rage
```

El atlas de tipos: las relaciones I

Utilizando relaciones (*sans objets*)

Ada

```ada
-- Ada 83
type    Grade   is range 0 .. 10;
subtype Failure is Grade range 0 .. 4; -- Compatible with Grade, but restricted range

type Math_Grade is new Grade;          -- Binary ideantical to Grade, incompatible
type Econ_Grade is new Grade;          -- Binary ideantical to Grade, incompatible

type Sports_Grade is new Grade range 1 .. 6; -- Binary identical to Grade,
                                             -- incompatible with all others
                                             -- restricted rage

type    Week      is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
subtype Work_Days is Days range Mon .. Fri;  -- Yes, enums have ranges too!
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```ada
-- Ada 83
Some_Grade : Grade   := 7;
Some_Fail  : Failure := 4; -- Reminder: a subtype

Some_Grade := Some_Fail;   -- Ok! It is always okay!
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```
-- Ada 83
Some_Grade : Grade   := 7;
Some_Fail  : Failure := 4; -- Reminder: a subtype

Some_Grade := Some_Fail;   -- Ok! It is always okay!

Some_Grade := 7;
Some_Fail  := Some_Grade:  -- Allowed by compiler as the types are compatible
                           -- Can cause runtime exception!!
                           -- In this case, it will!
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```ada
-- Ada 83
Some_Grade : Grade   := 7;
Some_Fail  : Failure := 4; -- Reminder: a subtype

Some_Grade := Some_Fail;   -- Ok! It is always okay!

Some_Grade := 7;
Some_Fail  := Some_Grade:  -- Allowed by compiler as the types are compatible
                           -- Can cause runtime exception!!
                           -- In this case, it will!

-- Ada 2012
Some_Fail := (if Some_Grade in Failure then Some_Grade else ...);
                            -- "in" is the most powerful keyword IMHO
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```ada
-- Ada 83
Some_Math_Grade : Math_Grade := 7;  -- Reminder: "new" type of Grade
Some_Econ_Grade : Econ_Grade := 7;  -- Reminder: "new" type of Grade
Some_Grade      : Grade      := 7;
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```
-- Ada 83
Some_Math_Grade : Math_Grade := 7;  -- Reminder: "new" type of Grade
Some_Econ_Grade : Econ_Grade := 7;  -- Reminder: "new" type of Grade
Some_Grade      : Grade      := 7;

Some_Econ_Grade := Some_Grade;      -- Compiler error!
                                    -- Incompatible types
Some_Math_Grade := Some_Econ_Grade; -- Compiler error!
                                    -- Incompatible types
```

El atlas de tipos: las relaciones II

Trabajando con relaciones de tipos

Ada

```
-- Ada 83
Some_Math_Grade : Math_Grade := 7;  -- Reminder: "new" type of Grade
Some_Econ_Grade : Econ_Grade := 7;  -- Reminder: "new" type of Grade
Some_Grade      : Grade      := 7;

Some_Econ_Grade := Some_Grade;      -- Compiler error!
                                    -- Incompatible types
Some_Math_Grade := Some_Econ_Grade; -- Compiler error!
                                    -- Incompatible types

Some_Math_Grade := Math_Grade(Some_Grade); -- Explicit casting needed
                                          -- No issues if range was not restricted
```

El atlas de tipos: lo atributos

Ada

```ada
-- Ada 83
Integer'First;         -- Smallest number
Integer'Last;          -- Largest  number
Integer'Succ(3);       -- Returns 4 (Successor   of 3)
Integer'Pred(3);       -- Returns 2 (Predecessor of 3)
Integer'Value("-10"); -- Returns -10
Boolean'Pos(False);    -- Normally 0, but not mandatory (specially in hardened systems)
Boolean'Val(0);        -- Would normally return False
My_Type'Size;          -- Size in _bits_ of the type!

My_Arr'Range           -- Returns the range of an array!
My_Var'Image;          -- Returns the string representation of the type
My_Var'Access;         -- Returns an access (pointer) to the variable
```

Ada 2022 RM K.2 Language-Defined Attributes

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
// C (old C++)
int My_Integer = -100;
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
// C (old C++)
const int MIN_VAL = -100;
int My_Integer    = MIN_VAL;
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
// C (old C++)
const int MIN_VAL = -100;
const int MAX_VAL =  100;
int My_Integer    =  MIN_VAL;
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
// C++11
struct My_Int_T {
    const int MIN_VAL = -100;
    const int MAX_VAL =  100;
          int value   =  MIN_VAL;
} My_Integer; // Still no checks...
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
Okay, now... for real, I promise...
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
// C++ 20!!
template<int min, int max>
struct test
{
    consteval test(int v): value(v) {
        if(v < min || v > max) throw v;
    }
    int value, min, max;
};
int main()
{
    test<-100, 100> t1(2);
    test<-100, 100> t2(120); // Comp-time error!
} // Still no runtime checks
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
///// Runtime checks??
// Old C++
#include<limits>
std::numeric_limits<int>::min();
// Generic & machine related types

// C++20
#include<utility>
std::in_range<some_type>(some_value);
// Generic types
#include<ranges>
std::range::range;
// Ranged type, but not a "simple" type
// C++26 contracts are not for types!!
```

Una primera parada, comparación con C++

Ada

```ada
-- Ada 83
type My_Int_T is range -100 .. 100;
My_Integer : My_Int_T := My_Int_T'First;
```

cpp

```cpp
///// Runtime checks??
// Old C++
#include<limits>
std::numeric_limits<int>::min();
// Generic & machine related types

// C++20
#include<utility>
std::in_range<some_type>(some_value);
// Generic types
#include<ranges>
std::range::range;
// Ranged type, but not a "simple" type
// C++26 contracts are not for types!!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Ada 2012 (imho with better syntax, but they existed since 95)
type My_Atomic_Num is digits 15 with Atomic, Default_Value => 0.0;
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- Ada 2012 (imho with better syntax, but they existed since 95)
type My_Atomic_Num is digits 15 with Atomic, Default_Value => 0.0;

type My_Smol_Enum is (Off, On) with Size => 1;  -- Of course we care about size
I2C1 : My_Smol_Enum with Address => 16#01000#;  -- Amazing for embedded
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- Ada 2012 (imho with better syntax, but they existed since 95)
type My_Atomic_Num is digits 15 with Atomic, Default_Value => 0.0;

type My_Smol_Enum is (Off, On) with Size => 1;  -- Of course we care about size
I2C1 : My_Smol_Enum with Address => 16#01000#;  -- Amazing for embedded

Smol_Var : Bit_1 with Address => I2C1'Address;  -- Memory overlays for the win!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Ada 2012 (imho with better syntax, but they existed since 95)
type My_Atomic_Num is digits 15 with Atomic, Default_Value => 0.0;

type My_Smol_Enum is (Off, On) with Size => 1;  -- Of course we care about size
I2C1 : My_Smol_Enum with Address => 16#01000#;  -- Amazing for embedded

Smol_Var : Bit_1 with Address => I2C1'Address;  -- Memory overlays for the win!

type My_Network  is xxxxx with Bit_Oder  => High_Order_First, -- Big Endian type!!
                               Alignment => 4;
                        -- for composite types, Scalar_Order may also be needed!!!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Ada 2012 (imho with better syntax, but they existed since 95)
type My_Atomic_Num is digits 15 with Atomic, Default_Value => 0.0;

type My_Smol_Enum is (Off, On) with Size => 1;  -- Of course we care about size
I2C1 : My_Smol_Enum with Address => 16#01000#;  -- Amazing for embedded

Smol_Var : Bit_1 with Address => I2C1'Address;  -- Memory overlays for the win!

type My_Network  is xxxxx with Bit_Oder  => High_Order_First, -- Big Endian type!!
                               Alignment => 4;
                        -- for composite types, Scalar_Order may also be needed!!!

type API1_Access is access API1_Type with Storage_Pool => API1_Storage_Pool;
                                       -- Yup, we also have pools/arenas for "pointers"
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- And there is so, so, so much more...
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
function MemCopy
  (Destination : System.Address;
   Source      : System.Address;
   Length      : Natural)
return Address
with
  Import,                                        -- Calling from other binaries
  Convention => C,                               -- It is C code convention calls
  Link_Name => "memcpy",                         -- Symbol name
  Pre  => Source /= Null_Address      and then -- Contract based programming!
          Destination /= Null_Address and then -- Including when calling to
          not Overlapping (Destination, Source, Length),
  Post => MemCopy'Result = Destination;          -- external functions!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);

type Increasing_Array is array (Index) of Some_Number
  with Dynamic_Predicate => (for all I in Index =>
    (if I < Index'Last then Increasing_Array(I) < Increasing_Array(I+1))); -- Nice!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```ada
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);

type Increasing_Array is array (Index) of Some_Number
  with Dynamic_Predicate => (for all I in Index =>
    (if I < Index'Last then Increasing_Array(I) < Increasing_Array(I+1))); -- Nice!
```

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);

type Increasing_Array is array (Index) of Some_Number
  with Dynamic_Predicate => (for all I in Index =>
    (if I < Index'Last then Increasing_Array(I) < Increasing_Array(I+1))); -- Nice!
```

Probablemente una de las cosas más chulas del lenguaje

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);

type Increasing_Array is array (Index) of Some_Number
  with Dynamic_Predicate => (for all I in Index =>
    (if I < Index'Last then Increasing_Array(I) < Increasing_Array(I+1))); -- Nice!
```

¡Los aspectos desconectan la implementación del problema!

El atlas de tipos: los aspectos

Aspectos de Tipos/Variables/etc

Ada

```
-- Formal specification/verification in Ada! Liquid types and contracts
-- ALL OF THIS IS ADA 2012!!!
type Even is new Natural with
  Dynamic_Predicate => Even mod 2 = 0; -- we write the actual properties of the data
type Odd  is new Natural with
  Dynamic_Predicate => not in Even; -- again, the "in" keyword is amazing!

type Prime is new Positive with
  Dynamic_Predicate => (for all Divisor in 2 .. Prime / 2 => Prime mod Divisor /= 0);

type Increasing_Array is array (Index) of Some_Number
  with Dynamic_Predicate => (for all I in Index =>
    (if I < Index'Last then Increasing_Array(I) < Increasing_Array(I+1))); -- Nice!
```

¡Los aspectos desconectan la implementación del problema!

El átlas de tipos: los tipos compuestos, arrays I

Arrays I

Ada

```ada
-- Ada 83
type My_Index is range -10 .. 10;
type Int_Arr  is array (My_Index) of Integer;
-- The Index can be any discrete type (Ints, Enums, Mods) and have any range

My_Array1 : Int_Arr := []; -- Uninitialised. Braces are from Ada 2022
My_Array2 : Int_Arr := [1, 2, 1, 2...]; -- Sequential initialization
My_Array3 : Int_Arr := [-10 | -8 | -6 ... => 1, -9 | -7 ... => 2]; -- Positional init
My_Array4 : Int_Arr := [others => 0]; -- Initialise all to 0
My_Array5 : Int_Arr := [-10 | -8 | -6 ... => 1, others => 2]; -- Mixed case

-- Ada 2022
My_Array6 : Int_Arr := [for I in My_Index'Range => Integer(My_Index)]; -- -10, -9...
My_Array7 : Int_Arr := [for I in 1 .. 3 => Integer(I), 5 .. 10 => 10, others => 0];
```

El átlas de tipos: los tipos compuestos, arrays II

Arrays II: índices indefinidos

Ada

```ada
-- Ada 83
type Int_Arr is array (Positive range <>) of Integer; -- "<>" means ellipsis

My_Int_Arr_Var  : Int_Arr(1 .. 10) := [others => 0];  -- bounds given explicitly
My_Int_Arr_Var2 : Int_Arr          := [1, 2, 3, 4];   -- bounds given by data
My_Int_Arr_Var3 : Int_Arr(1 ..  A) := [others => 0];  -- bounds given at runtime!

type Arr_Int_Arr is array (Positive range <>) of Int_Arr (1 .. 20);
                                                    -- ^bounds need to be given!!
-- VERY IMPORTANT!! The bounds/range of arrays are an INTRINSIC PART of the type!!!

type Bit_Mat is array (Integer range <>, Integer range <>) of Boolean;
Bit_Mat'Range; Bit_Mat'Range(2); Bit_Arr'Length; Bit_Arr'First; Bit_Arr'Last;
-- Guess what all these attributes do?
```

El atlas de tipos: los tipos compuestos II

Records I: *aka, structs*

Ada

```ada
-- Ada 83
type Data is record
   Day   : Integer range 1 .. 31;
   Month : Months;
   Year  : Integer range 1 .. 3000 := 2026; -- Default values allowed
end record; -- Aka, your typical struct in other languages

--  Positional components
Ada_Birthday    : Date := (10, December, 1815);
--  Named components
Leap_Day_2020   : Date := (Day   => 29,
                           Month => February,
                           Year  => 2020); -- Init by name
type Empty is null record; -- Empty "container", it is quite useful!
```

El atlas de tipos: los tipos compuestos II

Records II: discriminantes

Ada

```ada
-- Discriminated records: they are different/discriminated by their input
type Square_Mat (Size : Positive) is record       -- Unconstrained discriminant
      Mat : Matrix(1 .. Size, 1 .. Size);
end record;
type Buffer (Size : Buffer_Size := 100) is record -- Default initialized
   Pos   : Buffer_Size := 0;
   Value : String(1 .. Size);
end record;

Basis   : Square_Mat(5); -- constrained, always 5 by 5
ILLEGAL : Square_Mat;    -- illegal, a Square_Mat must be constrained

Large   : Buffer(200);   -- constrained, always 200 characters (explicit discriminant)
Message : Buffer;        -- _unconstrained_, initially 100 characters, but can change
                         -- (default discriminant value)
```

El atlas de tipos: los tipos compuestos II

Records III: variantes

Ada

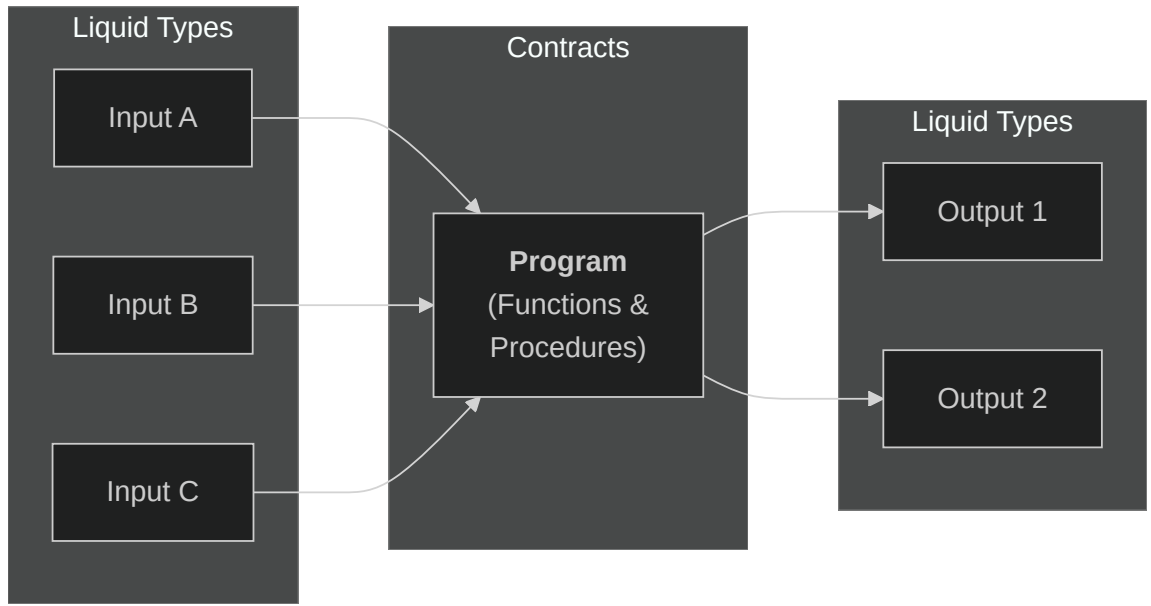
```
type Expr (Kind : Expr_Kind_Type) is record
                -- ^ The discriminant is an enumeration value
   Current_Val : Float; -- This member is here allways
   case Kind is -- Variant part
      when Bin_Op_Plus | Bin_Op_Minus => Left, Right : Operator;
      when Num => Val : Integer;
   end case;
end record; -- Very similar to C/C++/Rust union types

case E.Kind is
   when Bin_Op_Plus  => Eval_Expr (E.Left) + Eval_Expr (E.Right),
   when Bin_Op_Minus => Eval_Expr (E.Left) - Eval_Expr (E.Right),
   when Num          => E.Val;
end case;
```

¿Por qué son tan importantes los tipos de datos?

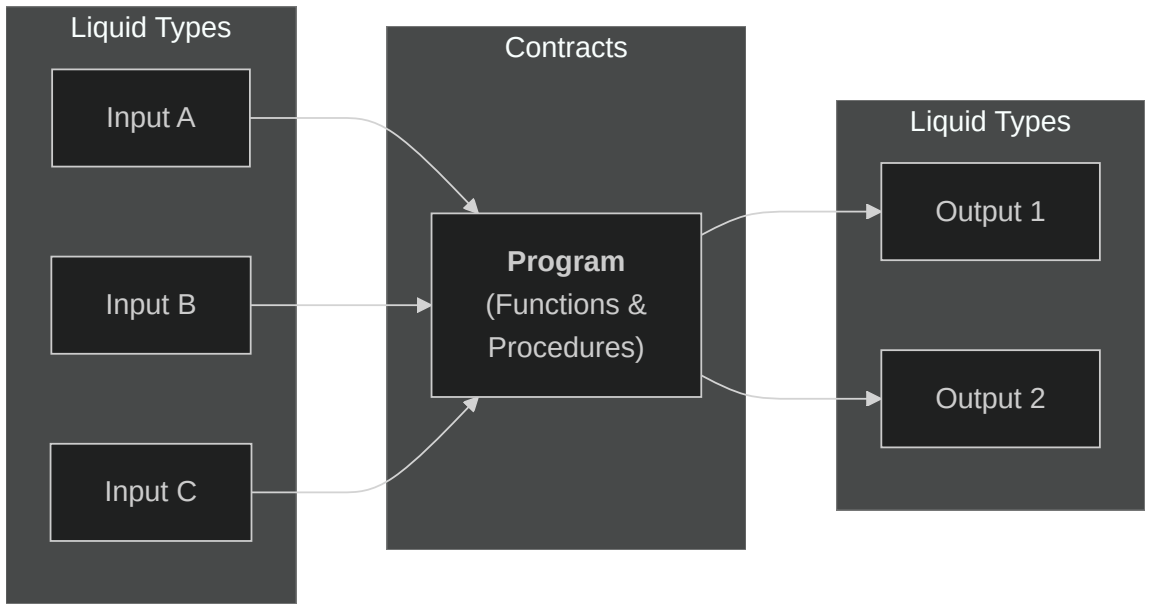
¿Por qué son tan importantes los tipos de datos?

Los datos son el "50%" del problema



¿Por qué son tan importantes los tipos de datos?

Los datos son el "50%" del problema



¡Los datos cuando están bien definidos, son la base para construir el resto!

¡Desarrolla los inputs y outputs, y el resto fluye por si mismo!

La estructura de un programa en Ada

Ada

```ada
with Ada.Text_IO; use Ada.Text_IO; -- Import libraries and "namespaces"

procedure Swap_Test is
   -- Declarative section. We can even have full-blown functions/procedures!
   procedure Swap (A, B: in out Integer) is  -- We indicate the flow of data!
      Tmp : constant Integer := A;  -- Initialization in declaration
   begin
      A := B;
      B := Tmp;
   end Swap;
   A : Integer := 10;
   B : Integer := 20;
begin
   Swap (A, B);
   Put_Line ("A: " & A'Image); Put_Line ("B: " & B'Image);
end Swap_Test;
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```ada
loop
  Val := Do_Something(Val);
  exit when Val > 10; -- Exit condition for the loop with the "exit when" clause
  ...
  Do_Something_Else();
end loop;
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```ada
Example:
 loop
   Val := Do_Something(Val);
   exit when Val > 10;
   ...
   Do_Something_Else();
 end loop Example;     -- Named loop. Very useful for nested loops
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```
Example:
 loop
   Val := Do_Something(Val);
   Another_Loop:
    loop
      exit Example when Val > 10; -- Exit condition for the outer loop!
      Some_Nested_Operation();
    end loop Another_Loop;
   ...
   Do_Something_Else();
 end loop Example;      -- Named loop. Very useful for nested loops
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```ada
for I in Class_Grades'Range loop -- Indexed-based looping
   if Class_Grades(I) in Failure then -- "I", the index, CANNOT BE CHANGED!!!
      ...
   end if;
end loop;
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```ada
for I in reverse Class_Grades'Range when Class_Grades(I) in Failure loop
     -- "reserve" is needed if we want to invert iterations
     -- Functional style loops
     ...
end loop;
```

Loops en Ada

Las pinceladas más básicas posibles...

Ada

```ada
for I in reverse Class_Grades'Range when Class_Grades(I) in Failure loop
     -- "reserve" is needed if we want to invert iterations
     -- Functional style loops
     ...
end loop;

for V of Class_Grades when V in Failure loop
  -- "of" is an iterator into the data
     ...
end loop;
```

Condicionales en Ada

Ada

```ada
-- =, /=, <, <=, >, >=, in, not in, and, and then, or, or else, xor
if V > 5 then
   ...
elsif V > 8 then
   ...
else
   ...
end if;
```

Condicionales en Ada

Ada

```ada
-- =, /=, <, <=, >, >=, in, not in, and, and then, or, or else, xor
if V > 5 then
   ...
elsif V > 8 then
   ...
else
   ...
end if;

case Sensor is -- Exhaustive matching in mandatory!
   when Elevation => Record_Elevation(Sensor_Value);
   when Azimuth   => Record_Azimuth  (Sensor_Value);
   when Distance  => Record_Distance (Sensor_Value);
   when others    => null;
end case;
```

Condicionales en Ada

Ada

```ada
-- =, /=, <, <=, >, >=, in, not in, and, and then, or, or else, xor
if V > 5 then
   ...
elsif V > 8 then
   ...
else
   ...
end if;

case Bin_Number(Count) is -- For numerics, "when others" is required
   when 1         => Update_Bin(1);
   when 2         => Update_Bin(2);
   when 3 | 4     => Empty_Bin(1);
                     Empty_Bin(2);
   when others    => raise Error;
end case;
```

Tareas en Ada

La menor pincelada que vayáis a ver

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Simple_Tasks is
   task Task1; task Task2;
   task body Task1 is
   begin
      loop
         Put (". "); delay 0.5;
      end loop;
   end Task1;
   task body Task2 is
   begin
      loop
         Put ("# "); delay 1.0;
      end loop;
   end Task2;
begin
   Put ("In main! ");
end Simple_Tasks; -- We don't terminate!?
```

Tareas en Ada

La menor pincelada que vayáis a ver

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Simple_Tasks is
   task Task1; task Task2;
   task body Task1 is
   begin
      loop
         Put (". "); delay 0.5;
      end loop;
   end Task1;
   task body Task2 is
   begin
      loop
         Put ("# "); delay 1.0;
      end loop;
   end Task2;
begin
   Put ("In main! ");
end Simple_Tasks; -- We don't terminate!?
```

```
. # In main! . # . . # . . # . . # . . # . . # . ^C
```

Tareas en Ada

La menor pincelada que vayáis a ver

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Simple_Tasks is
   task Task1; task Task2;
   task body Task1 is
   begin
      loop
         Put (". "); delay 0.5;
      end loop;
   end Task1;
   task body Task2 is
   begin
      loop
         Put ("# "); delay 1.0;
      end loop;
   end Task2;
begin
   Put ("In main! ");
end Simple_Tasks; -- We don't terminate!?
```

```
. # In main! . # . . # . . # . . # . . # . . # . ^C
```

- **¡No hay que importar librerías!**
- ¡*Joined* por defecto! (*main* espera)
- Se ejecutan nada más estar disponibles
  - Minimiza carga cognoscitiva y aumenta rendimiento
- Obviamente las `task` tienen montones y montones más por detrás

SPARK

El músculo de la programación

Razonando con la información y el flujo de datos

Usando todo el trabajo hecho, para un resultado excelente

SPARK es, hoy en día, Ada, y viceversa

```
function P_Positions_Swapped  -- From SPARKlib
  (Left : P.Map; Right : P.Map; X : Cursor; Y : Cursor) return Boolean
with
  Global => null,
  Post   =>
    (SPARKlib_Full =>
       P_Positions_Swapped'Result
       = (P.Same_Keys (Left, Right)
          and P.Elements_Equal_Except (Left, Right, X, Y)
          and P.Has_Key (Left, X)
          and P.Has_Key (Left, Y)
          and P.Get (Left, X) = P.Get (Right, Y)
          and P.Get (Left, Y) = P.Get (Right, X)));
--  Left and Right contain the same cursors, but the positions of X and Y are reversed
```

Razonando con la información y el flujo de datos

Usando todo el trabajo hecho, para un resultado excelente

SPARK es, hoy en día, Ada, y viceversa

```
function Count_Rec  -- From SPARKlib
  (S    : Sequence;
   Last : Extended_Index;
   Test : not null access function (E : Element_Type) return Boolean)
   return Big_Natural
with
  Ghost              => SPARKlib_Full,
  Subprogram_Variant => (Decreases => Last),
  Pre                => Last <= Vectors.Last (S),
  Post               =>  Count_Rec'Result <= Big (Last) - Big (Extended_Index'First);
--  Recursive version of Count
```

Razonando con la información y el flujo de datos

Usando todo el trabajo hecho, para un resultado excelente

SPARK es, hoy en día, Ada, y viceversa

```
function Count_Rec  -- From SPARKlib
  (S    : Sequence;
   Last : Extended_Index;
   Test : not null access function (E : Element_Type) return Boolean)
   return Big_Natural
with
  Ghost              => SPARKlib_Full,
  Subprogram_Variant => (Decreases => Last),
  Pre                => Last <= Vectors.Last (S),
  Post               =>  Count_Rec'Result <= Big (Last) - Big (Extended_Index'First);
--  Recursive version of Count
```

Errores de aritmética, memoria, flujo, propiedades, rangos, especificaciones, requisitos...
Usa métodos formales SMT (Z3, CVC5, Alt-ergo...). **SPARK lo comprueba TODO.**

Conclusiones

That is all (for today) folks!

El valor de Ada/SPARK

El trabajo de un programador

- Programar
- Reescribiendo código
- Debuggear
- Leer código
- Leer documentación
- Escribir documentación
- Crear tests
- Procesos lentos de verificación
- Crear herramientas ancilares
- Discusiones y reuniones

¿Y la IA?

El valor de Ada/SPARK

El trabajo de un programador

- Programar
- Reescribiendo código
- Debuggear
- Leer código
- Leer documentación
- Escribir documentación
- Crear tests
- Procesos lentos de verificación
- Crear herramientas ancilares
- Discusiones y reuniones

¿Y la IA?

Ada mejora todos los aspectos

- ...
- Mayor apoyo para restructuraciones
- Menos bugs, detección más fácil
- Más fácil leer el código
- Menos lectura de documentación
- Menos necesidad de documentación
- Mínima o ninguna necesidad de tests
- Verificación de sistemas integrado
- Entorno cohesivo de herramientas
- Menos comunicaciones y errores

¡Comprobar, modificar código y mantener la calidad con IA es trivial!

No todo lo que reluce es plata...

Hay muy pocos trabajos de Ada

- La mayoría requieren mucha experiencia y se comunican en círculos cerrados.

No todo lo que reluce es plata...

Hay muy pocos trabajos de Ada

- La mayoría requieren mucha experiencia y se comunican en círculos cerrados.
- ¡Pero aprender Ada nunca ha sido un negativo, lo contrario!

No todo lo que reluce es plata...

Hay muy pocos trabajos de Ada

- La mayoría requieren mucha experiencia y se comunican en círculos cerrados.
- ¡Pero aprender Ada nunca ha sido un negativo, lo contrario!

La comunidad de Ada es muy pequeña

- Hay muy pocas librerías...
- Algunas herramientas son muy bajas...

No todo lo que reluce es plata...

Hay muy pocos trabajos de Ada

- La mayoría requieren mucha experiencia y se comunican en círculos cerrados.
- ¡Pero aprender Ada nunca ha sido un negativo, lo contrario!

La comunidad de Ada es muy pequeña

- Hay muy pocas librerías...
- Algunas herramientas son muy bajas...
- ¡Pero eso es una oportunidad si queréis haceros un nombre!
- Y la comunidad da bastante apoyo técnico

Proyectos que merecen mucho la pena

OSes (y SPARK)

- Ironclad/Gloire: UNIX-like OS kernel (parcialmente) verificado, capaz de RT, de uso general y empotrado escrito en Ada/SPARK
- Muen: *A Separation Kernel for High Assurance (Ada/SPARK)*

SPARK

- SPARKlib: librería estándar en 100% SPARK
- SPARKNaCl: TweetNaCl en SPARK
- RecordFlux: generador de parsers, generadores, protocolos formalmente verificados

Empotrado y utilidades

- SweetAda: framework para correr Ada en "cualquier dispositivo"
- GHDL: compilador de VHDL escrito en Ada
- Simple Components: enorme librería de utilidades en Ada. Excelente recurso.

La comunidad Ada-Lang y el foro!

Debajo de la página tenéis links para los grupos de Telegram, Matrix, Discord

GNAT Academic Program (GAP) (interesante para la Uni)

Asociaciones

- Ada-Spain
  - ¡Hay un premio cada dos años!
- Ada User Society
  - ¡Se van a publicar recompensas por trabajos! Ver el anuncio
- Ada-Europe

Historicamente UseNet y #ada

Ada/SPARK en 3 minutos...

Alire package manager (*ala* `cargo`)

Alire te instala el compilador (nativo y cross) y GNATprove/SPARK

```sh
alr init --bin new_project # Initialise a project
cd new_project

alr with sparklib # Install dependencies

alr edit # Edit your application, Alire setsup the LSP and the like!

alr build # Build the applications

alr gnatprove # Run the prover
```

Ver también GNAT Studio

Aprende y prueba Ada/SPARK en el navegador

No hemos visto ni el 25% del lenguaje... Hay cosas chulísimas que ni he mencionado

Learn.AdaCore

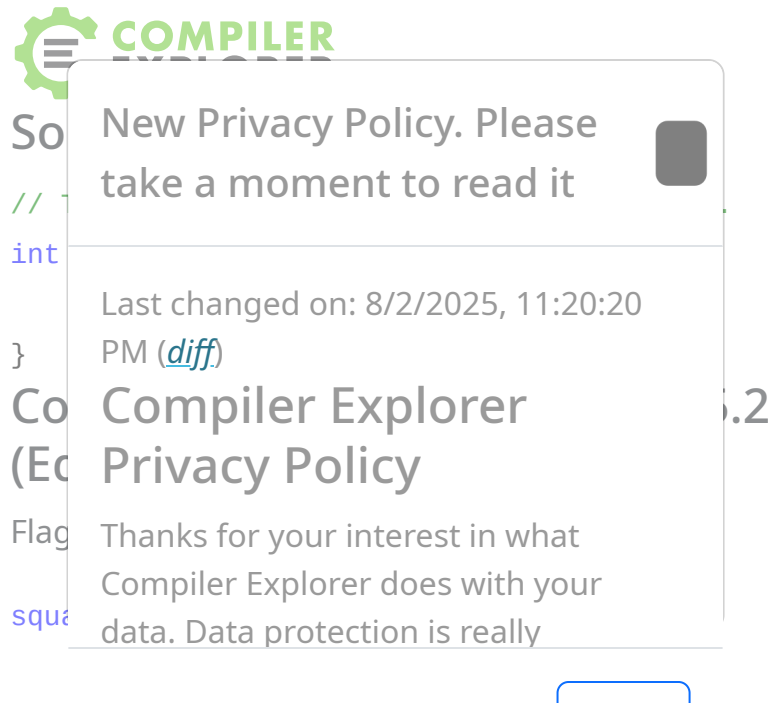
LEARN.ADACORE.COM

Edit on GitHub

What is Ada and SPARK?

Ada is a state-of-the art programming language

Compiler Explorer/Godbolt



Gracias a...

- Mona Sans & Monaspace Neon por la tipografía
- La comunidad de Ada
- El motor de presentaciones Sli.dev
- A toda la comunidad de software libre y el propio software
- Izán y la asociación por haber propuesto y organizado la charla

Recursos extra - Videos

- The Outsider's Guide to Ada - Paul Jarret
- An Introduction to Ada for Beginning and Experienced Programmers - Jean-Pierre Rosen
- Understanding liquid types, contracts and formal verification with Ada/SPARK - Fernando Oleo Blanco
- Tracking Performance of a Big Application from Dev to Ops - Philippe Waroquiers
- Is Ada memory safe? - Reddit, Niklas Holsti

¡Gracias!

¿Preguntas?

Fernando Oleo Blanco
https://irvise.xyz/
irvise@irvise.xyz