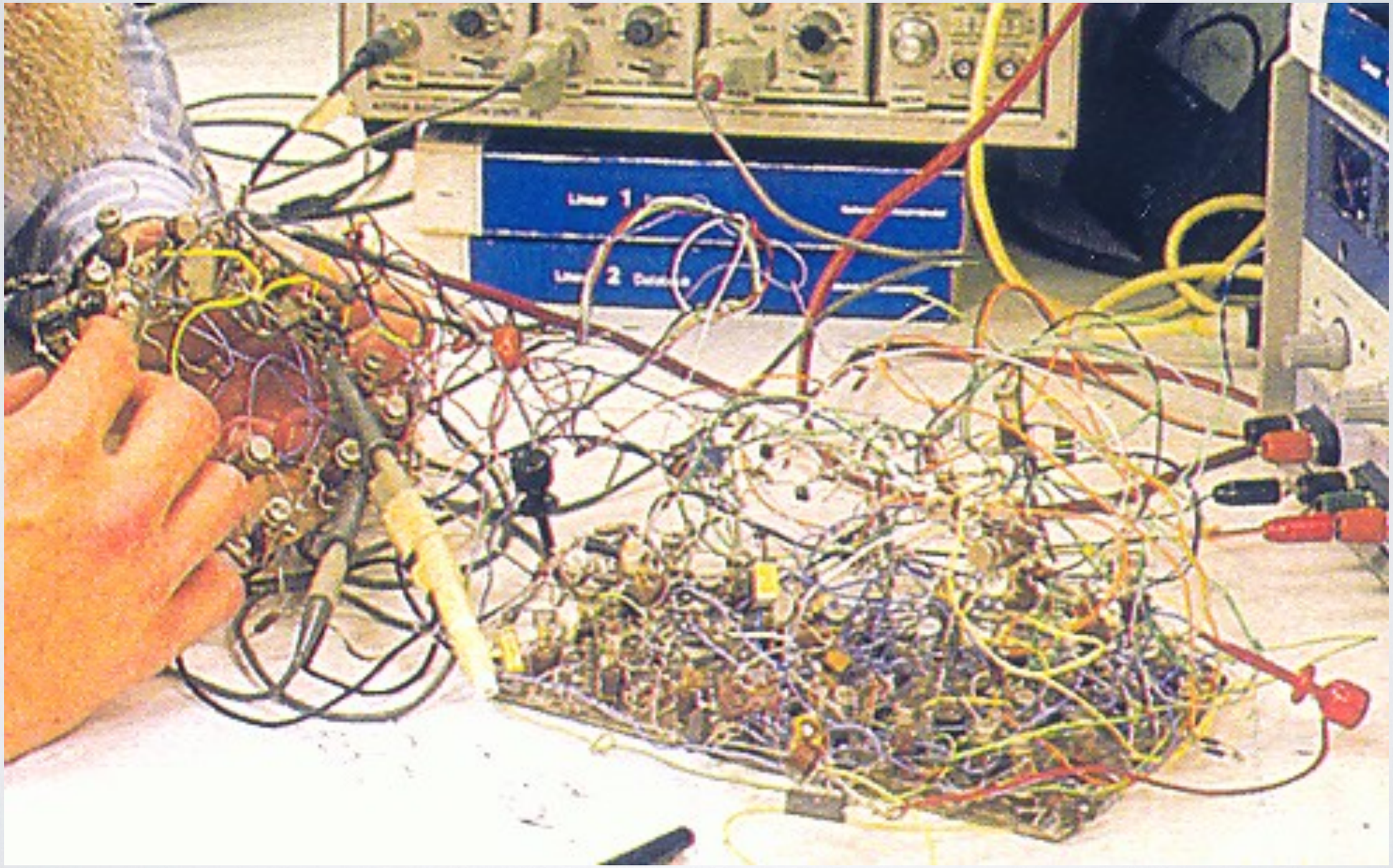
WPROWADZENIE

DLACZEGO ADA?



Obraz zaczerpnięty ze strony https://www.electronicdesign.com

GDZIE JEST STOSOWANA ADA

RAPITA SYSTEMS
A DANLAW COMPANY

AIRBUS

BOEING

ROLLS ROYCE

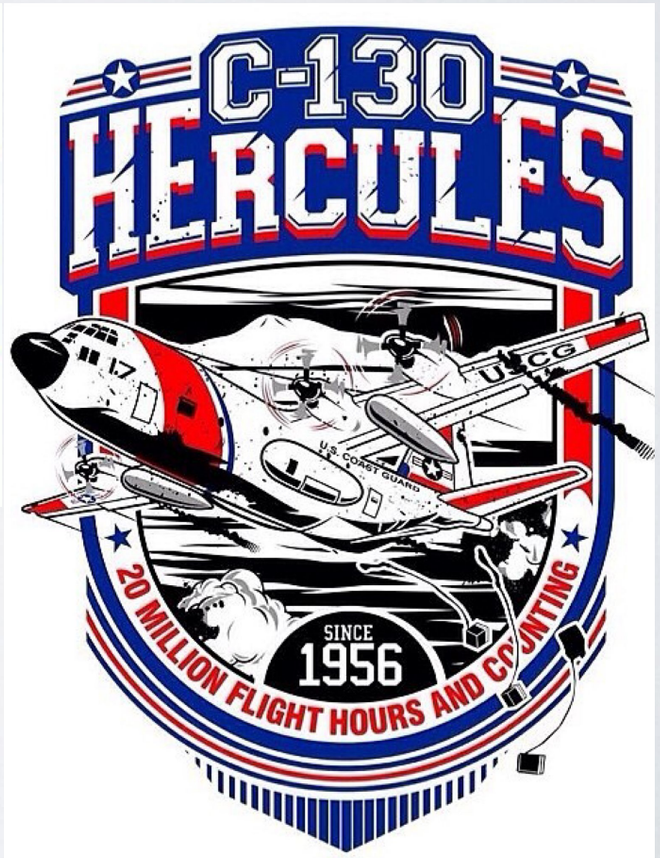
BNP PARIBAS

JEOL

ROS

instytutlotnictwa
warszawa, rok założenia 1926

Hertz



LOCKHEED MARTIN

THALES

Loga zaczerpnięte ze strony producentów

HISTORIA I WERSJE ADY

1983

DEPARTMENT OF DEFENSE
REQUIREMENTS FOR HIGH ORDER
COMPUTER PROGRAMMING LANGUAGES

"STEELMAN"

June 1978

PREFACE

The Department of Defense Common High Order Lar
1975 with the goal of establishing a single high order
propriate for DoD embedded computer systems. A
(HOLWG) was established to formulate the DoD
to evaluate existing languages against those r
set of languages required for DoD use. As a
goal, DoD Directive 5000.29 provides
grammed in a DoD approved and centra
5000.31 gives an interim list of apr
CMS-2, SPL/1, and JOVIAL J3 and
benefits from increased use of hig
of a single modern language v
have been widely distributed
producing successively m
MAN, TINMAN, IRON
velopment process, it
both necessary and
formed on dozen
adopted as a sin
all the requir
duce comp
of desig
will er
prod
of the l
tools, as we
and well maint

THE TECHNICAL REQUIREMENTS

The technical requirements for a common DoD high order progra
here are a synthesis of the requirements submitted by the Military I
ify a set of constraints on the design of languages that are approp
puter applications (i.e., command and control, communications
equipment, software development and maintenance, and suppo
especially like to thank the phase one analysis teams, the lan
many other individuals and organizations that have commer
and have identified weaknesses and trouble spots in the tech
goal in this revision has been to reduce the complexity of th

This revision incorporates the following changes. Care h
paragraph numbers remain the same as in the Revised I
changes in terminology and many changes in wording to
preciseness of the requirements. Several requirements
straints that were unintended but were implied becaus
ular mechanism rather than giving the underlying
embedded comments (2I), unordered enumeration t
ifications (7D), dynamic aliasing of array compon
of data (11B) have been deleted because they hav
quately justified. The minimal source language c
acters to make it compatible with the majorit
together model for parallel processing has bee
applications and has been replaced by a requir
preliminary designs have demonstrated the
conversion between types (3B), subtype con
tinction between open and closed scopes (
mechanisms, to pass data between parallel
(10F), to wait for several signals simultan

The Steelman is organized with an outl
document. Section 1 gives the genera
influenced the selection of more spec
for language design decisions that a
2 through 12 give more specific co
man calls for the inclusion of featu
tion, and maintenance of mil
characteristics desired for the language
characteristics. Section 13 gives some of the im
control, and use of the language. The intended use and
strongly influenced the requirements, and should influence the lan

A precise and consistent use of terms has been attempted throu
potentially ambiguous terms have been defined in the

1. General Design Criteria

1A. Generality. The language shall provide gene
satisfy the needs of embedded computer appl
time control, self diagnostics, input-output
processing, numeric computation, and file processing.

1B. Reliability. The language should aid the design and development of reliable pro-
grams. The language shall be designed to avoid error prone features and to maximize
automatic detection of programming errors. The language shall require some redun-
dant, but not duplicative, specifications in programs. Translators shall produce explan-
atory diagnostic and warning messages, but shall not attempt to correct programming
errors.

1C. Maintainability. The language should promote ease of program maintenance. It
should emphasize program readability (i.e., clarity, understandability, and modifiabil-
ity of programs). The language should encourage user documentation of programs. It
shall require explicit specification of programmer decisions and shall provide defaults
only for instances where the default is stated in the language definition, is always
meaningful, reflects the most frequent usage in programs, and may be explicitly over-
ridden.

1D. Efficiency. The language design should aid the production of efficient object pro-
grams. Constructs that have unexpectedly expensive implementations should be chosen to have a simple
recognizable by translators and by users. Features should be chosen to have a simple
and efficient implementation in many object machines, to avoid execution costs for
available generality where it is not needed, to maximize the number of safe optimiza-
tions available to translators, and to ensure that unused and constant portions of pro-
grams will not add to execution costs. Execution time support packages of the
language shall not be included in object code unless they are called.

1E. Simplicity. The language should not contain unnecessary complexity. It should
have a consistent semantic structure that minimizes the number of underlying con-
cepts. It should be as small as possible consistent with the needs of the intended
applications. It should have few special cases and should be composed from fe
that are individually simple in their semantics. The language should have uni
tactic conventions and should not provide several notations for the same
arbitrary restriction should be imposed on a language feature.

1F. Implementability. The language shall be composed from
stood and can be implemented. The semantics of each fea
well specified and understandable that it will be possibl
other features. To the extent that it does not interfere
guage shall facilitate the production of translators
efficient during translation. There shall be
forceable by translators.

HISTORIA I WERSJE ADY

1983

Cechy charakterystyczne wersji:

- Obsługa wyjątków
- Pakiety
- Prywatne typy
- Bardzo silne wyróżnione typy liczbowe
- Całkowicie statyczny język (brak rozszerzeń typów, brak parametrów w procedurach, brak polimorfizmów działających w programie itd.)

HISTORIA I WERSJE ADY

1995

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION

INFORMATION TECHNOLOGY -- PROGRAMMING LANGUAGES -- ADA

[Revision of first edition (ISO 8652:1987)]

ADA REFERENCE MANUAL

Language and Standard Libraries

Version 6.0
21 December 1994

Cechy charakterystyczne wersji:

- Chronione obiekty
- Hierarchiczne biblioteki
- Object oriented programing
- Polimorfizmów i dziedziczeń
- Tasków i typów generycznych
- Pierwszy na świecie zestandaryzowany język obiektowy

Źródła zaczerpnięte ze strony https://www.adahome.com

HISTORIA I WERSJE ADY

2005

Cechy charakterystyczne wersji:

- Interfejsy (enkapsulacja)
- Kontenery (nie do końca dopracowane)
- List, wektorów, Map, setów.
- Poprawa funkcjonalności (tasków i podejścia obiektowego)
- Podniesienie poziomu abstrakcji

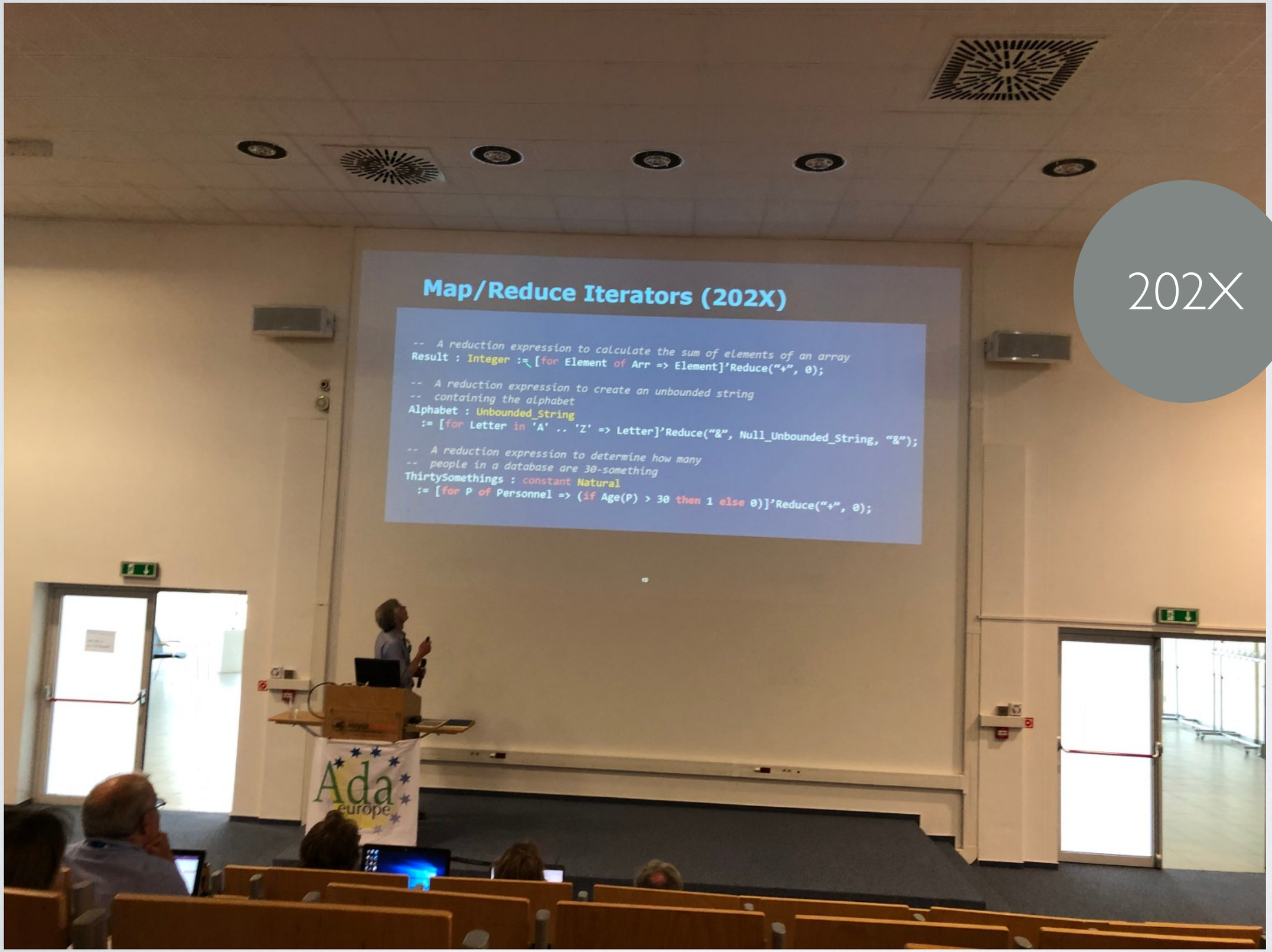
HISTORIA I WERSJE ADY

2012

Cechy charakterystyczne wersji:

- Programowanie przez kontrakt
- Wysoko poziomowe iteratory dla kontenerów
- Nowe wyrażenia warunkowe
- Nowe wyrażenia ilościowe

HISTORIA I WERSJE ADY



NARZĘDZIA BIBLIOTEKI FORA

- https://www.adaic.org
- https://www.adacore.com/download
- http://getadanow.com
- http://rosettacode.org/wiki/Category:Ada
- https://stackoverflow.com/questions/tagged/ada
- https://github.com/AdaCore
- https://github.com/AdaCore/Ada_Drivers_Library
- https://www.makewithada.org
- https://twitter.com/adaprogrammers
- https://learn.adacore.com

ADA W SYSTEMACH EMBEDDED

Make with Ada PROGRAMMING COMPETITION

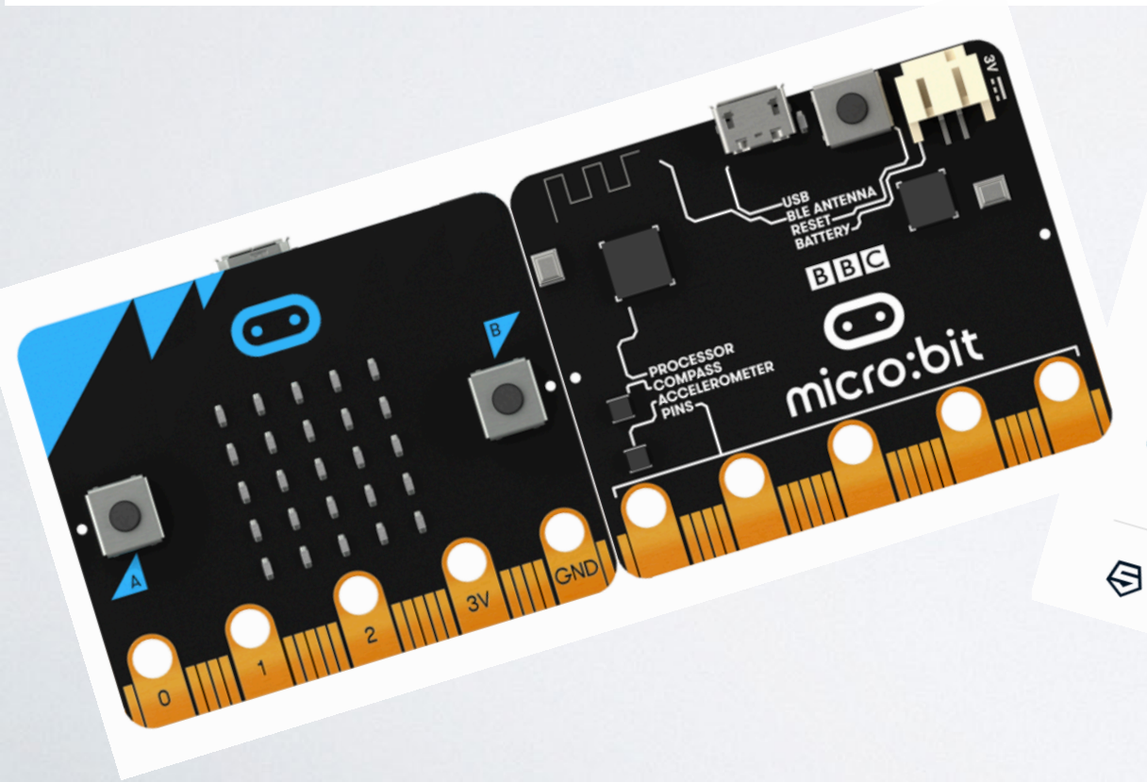
September 10, 2019 — January 31, 2020

Register at Hackster »

Make with Ada is an embedded software project competition sponsored by AdaCore. It is open to individuals and small teams using the Ada or SPARK languages to develop dependable, open, inventive and collaborative projects.

EACH OF THE TOP TEN FINALISTS WILL RECEIVE **$600**

FIRST PLACE GETS AN ADDITIONAL **$2K**



What are the supported platforms?

To start your project you will probably need some hardware. Below are some platforms that a
supported. Of course, adding support for another platform would be a great contribution for th

Board	Architecture	Where to buy	Where
BBC Micro:Bit	ARM Cortex-M0	• AdaFruit (US) • Pimoroni (UK/EU) • Farnell	Ada_Driv
STM32 by STM32F4 Discovery	ARM Cortex-M4F	• Mouser • Farnell • DigiKey	Ada_Drivers_
STM32 by STM32F429 Discovery	ARM Cortex-M4F	• Mouser • Farnell • Digikey	Ada_Drivers_Libr
STM32 by STM32F469 Discovery	ARM Cortex-M4F	• Mouser • Farnell • Digikey	Ada_Drivers_Library
STM32 by STM32F746 Discovery	ARM Cortex-M7F	• Mouser • Farnell • Digikey	Ada_Drivers_Library
Crazyflie 2.0 nano drone	ARM Cortex-M4F	• Bitcraze • Seeedstudio • Hackaday	The Certyflie project
HiFive1	RISC-V	• Crowd Supply	

ADA VS C/C++
Standardowa struktura kodu w pliku źródłowym - ADA

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  begin
6     Put_Line ("Good Morning");
7
8  end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./learn
  Good Morning
```

ADA VS C/C++
Fixed point … - ADA

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  type BYTE is mod 2**8; -- 8 bit
6
7  tmp : Integer; -- 32 bit
8  tmp_1, tmp_2, tmp_3 : Integer := 10;
9
10 tmp_4 : constant Integer := 2#1111_0101_0000_0001#;
11
12 tmp_5, tmp_6 : Integer := 1_100_000;
13 tmp_7 : Short_Integer := 16#7FCD#; --16 bit
14
15 tmp_8 : BYTE := -127;
16
17 text : Character := 'X';
18
19 flag : Boolean := TRUE;
20
21 begin
22    Put_Line ("Good Morning")
23
24 end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:7:01: warning: variable "tmp" is never read and never assigned
  learn.adb:8:01: warning: variable "tmp_1" is not referenced
  learn.adb:8:08: warning: variable "tmp_2" is not referenced
  learn.adb:8:15: warning: variable "tmp_3" is not referenced
  learn.adb:10:01: warning: constant "tmp_4" is not referenced
  learn.adb:12:01: warning: variable "tmp_5" is not referenced
  learn.adb:12:08: warning: variable "tmp_6" is not referenced
  learn.adb:13:01: warning: variable "tmp_7" is not referenced
  learn.adb:15:01: warning: variable "tmp_8" is not referenced
  learn.adb:17:01: warning: variable "text" is not referenced
  learn.adb:19:01: warning: variable "flag" is not referenced
$ ./learn
  Good Morning
```

ADA VS C/C++
Floating point & records - ADA

learn.adb

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

procedure Learn is

--Fixed point
type Temp_t is delta 0.0001 range -1.0 .. 1.0;
type Voltage_t is delta 0.01 digits 5;
-- Floating point
type Value_t is digits 3 range 0.0 .. 10.0E1;

tmp_1 : Temp_t := -0.67;
tmp_2 : Voltage_t := 123.71;
tmp_3 : Value_t := 9.91;

begin
   Put_Line ("Value");
   Put_Line (tmp_1'Image);
   Put_Line (tmp_2'Image);
   Put_Line (tmp_3'Image);

end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:11:01: warning: "tmp_1" is not modified, could be declared constant
  learn.adb:11:20: warning: static fixed-point value is not a multiple of Small
  learn.adb:12:01: warning: "tmp_2" is not modified, could be declared constant
  learn.adb:13:01: warning: "tmp_3" is not modified, could be declared constant
$ ./learn
  Value
  -0.6700
  123.71
  9.91E+00
```

ADA VS C/C++
Floating point & records - ADA

learn.adb

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

procedure Learn is

type d_float is delta 0.01 digits 3;
type my_array is array (0 .. 16) of Integer;

type number_struct is
   record
      tmp : d_float := -9.99;
      tmp_1 : my_array := (others => 7);
   end record;
type number_t is new number_struct;

number : number_t;

begin

   Put_Line ("Values:");
   Put_Line (number.tmp_1(1)'Image);
   Put_Line (number.tmp_1(16)'Image);
   Put_Line (number.tmp'Image);
   number.tmp := 0.00;
   Put_Line (number.tmp'Image);
   number.tmp := 0.01;
   Put_Line (number.tmp'Image);
   number.tmp := 9.99;
   Put_Line (number.tmp'Image);

end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
$ ./learn
  Values:
  7
  7
  -9.99
  0.00
  0.01
  9.99
```

ADA VS C/C++
Arrays - ADA

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  tmp : constant array (0 .. 3) of Integer := (1, 2, 3, 4);
6  tmp_1 : array (1 .. 4, 0 .. 16) of Short_Integer;
7
8  element : Integer := 0;
9
10 begin
11    element := tmp'Last;
12
13    Put_Line ("Value of last element:");
14    Put_Line (element'Image);
15
16    element := tmp'First;
17    Put_Line ("Value of first element:");
18    Put_Line (element'Image);
19
20    element := tmp(tmp'Last);
21    Put_Line ("Value of last element:");
22    Put_Line (element'Image);
23
24    element := tmp(tmp'First);
25    Put_Line ("Value of first element:");
26    Put_Line (element'Image);
27
28 end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:6:01: warning: variable "tmp_1" is never read and never assigned
$ ./learn
  Value of last element:
  3
  Value of first element:
  0
  Value of last element:
  4
  Value of first element:
  1
```

ADA VS C/C++
Arrays - ADA

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  tmp_1 : array (0 .. 16) of Integer := (1, 2, 3, 4, others => 0);
6
7
8  begin
9
10    Put_Line ("Values of before:");
11    Put_Line (tmp_1(13)'Image);
12    Put_Line (tmp_1(14)'Image);
13    Put_Line (tmp_1(15)'Image);
14    Put_Line (tmp_1(16)'Image);
15
16    tmp_1(13 .. 16) := tmp_1(0 .. 3);
17
18    Put_Line ("Values of after:");
19    Put_Line (tmp_1(13)'Image);
20    Put_Line (tmp_1(14)'Image);
21    Put_Line (tmp_1(15)'Image);
22    Put_Line (tmp_1(16)'Image);
23
24 end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:5:01: warning: variable "tmp" is not referenced
$ ./learn
  Values of before:
  0
  0
  0
  0
  Values of after:
  1
  2
  3
  4
```

ADA VS C/C++
Regs/bit field- ADA
Direct access to memory - ADA

learn.adb

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

procedure Learn is

type dtc_t is
   record
      tmp_1  : Boolean;
      tmp_2  : Natural range 0 .. 3;
      tmp_3  : Natural range 0 .. 2;
   end record;

for dtc_t use
   record
      tmp_1  at 0 range 0 .. 0;
      tmp_2  at 0 range 1 .. 4;
      tmp_3  at 0 range 5 .. 7;
   end record;
dtc : dtc_t;

begin
   dtc.tmp_1 := TRUE;

   Put_Line (dtc.tmp_1'Image);
   Put_Line (dtc.tmp_2'Image);
   Put_Line (dtc.tmp_3'Image);

   dtc.tmp_2 := 2#0010#;

   Put_Line (dtc.tmp_1'Image);
   Put_Line (dtc.tmp_2'Image);
   Put_Line (dtc.tmp_3'Image);
end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
$ ./learn
 Values:
TRUE
0
0
 Values:
TRUE
2
0
```

ADA VS C/C++
Discriminated types & Variant records - ADA

learn.adb

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

procedure Learn is

type ecu_type_t is (Engine, Hud, Radio);

type dtc_t (ecu_type : ecu_type_t) is
   record
      dtc_id : Integer;
      dtc_info : string(1 .. 15);
      case ecu_type is
      when Engine =>
         fatal_dtc_code : Long_Integer;
      when Hud =>
         mid_dtc_code : Integer;
         flag : Boolean;
      when Radio =>
         on_off : Boolean;
      end case;
   end record;

my_dtc_engine  : dtc_t(Engine);
my_dtc_hud     : dtc_t(Hud);
my_dtc_radio   : dtc_t(Radio);

begin
   my_dtc_engine.fatal_dtc_code := 99999;
   my_dtc_hud.mid_dtc_code := 12345;
   my_dtc_radio.on_off := TRUE;

   Put_Line ("Values:");
   Put_Line (my_dtc_engine.fatal_dtc_code'Image);
   Put_Line (my_dtc_hud.mid_dtc_code'Image);
   Put_Line (my_dtc_radio.on_off'Image);

end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
$ ./learn
  Values:
  99999
  12345
  TRUE
```

ADA VS C/C++
Discriminated types & Variant records - ADA

learn.adb

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

procedure Learn is

type dtc_log_t (Max_size : Integer) is
   record
      Nbr_dtc : Integer;
      dtc : string (1 .. Max_size);
   end record;

dtc_log : dtc_log_t(100);

begin

   dtc_log.dtc(97) :='W';

   Put_Line ("Values:");
   Put_Line (dtc_log.dtc(97)'Image);

end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./learn
  Values:
  'W'
```

ADA VS C/C++
Access type (pointers) - ADA

learn.adb

```
 1  with Ada.Text_IO; use Ada.Text_IO;
 2
 3  procedure Learn is
 4
 5  type Tab_t is array (0 .. 15) of Integer;
 6  type Tab_access_t is access all Tab_t;
 7
 8  Tab_1 : aliased Tab_t := (others => 2);
 9  Tab_2 : aliased Tab_t := (others => 7);
10  Tab_1_access : Tab_access_t;
11  Tab_2_access : Tab_access_t;
12
13  begin
14     Tab_1_access := Tab_1'Access;
15     Tab_2_access := Tab_2'Access;
16
17     Put_Line ("Value");
18     Put_Line (Tab_1(7)'Image);
19     Put_Line (Tab_2(7)'Image);
20
21     Tab_1_access.all := Tab_2_access.all;
22
23     Put_Line ("Value");
24     Put_Line (Tab_1(7)'Image);
25     Put_Line (Tab_2(7)'Image);
26
27  end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
$ ./learn
Value
2
7
Value
7
7
```

ADA VS C/C++
Access type (pointers) - ADA

learn.adb

```
 1  with Ada.Text_IO; use Ada.Text_IO;
 2
 3  procedure Learn is
 4
 5  type Tab_t is array (0 .. 15) of Integer;
 6  type Tab_access_t is access all Tab_t;
 7
 8  Tab_1 : aliased Tab_t := (others => 2);
 9  Tab_2 : aliased Tab_t := (others => 7);
10  Tab_1_access : Tab_access_t;
11  Tab_2_access : Tab_access_t;
12
13  begin
14     Tab_1_access := Tab_1'Access;
15     Tab_2_access := Tab_2'Access;
16
17     Put_Line ("Value");
18     Put_Line (Tab_1(7)'Image);
19     Put_Line (Tab_2(7)'Image);
20
21     Tab_1_access(7) := 8;
22     Tab_2_access(7) := 9;
23
24     Put_Line ("Value");
25     Put_Line (Tab_1(7)'Image);
26     Put_Line (Tab_1(6)'Image);
27     Put_Line (Tab_2(7)'Image);
28     Put_Line (Tab_2(6)'Image);
29
30  end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
$ ./learn
 Value
 2
 7
 Value
 8
 2
 9
 7
```

ADA VS C/C++
Typing - C

main.c

```c
#include <stdio.h>

typedef int All_Numbers;

All_Numbers Number = 11;
int Int_data = 20;

int main()
{
    Number = Int_data;

    printf("Today is \n\r");
    printf ("%d", Number);

    return 0;
}

```

input

```
Today is
20
```

ADA VS C/C++
Strong typing - ADA

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5     type All_Numbers is new Integer range 10 .. 56;
6
7     Number : All_Numbers := 11;
8     Int : Integer := 20;
9
10 begin
11
12    Number := Int;
13
14    Put_Line ("Today is");
15    Ada.Text_IO.Put_Line (Number'Image);
16
17 end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:12:14: expected type "All_Numbers" defined at line 5
  learn.adb:12:14: found type "Standard.Integer"
  gprbuild: *** compilation phase failed
  Build failed...
```

ADA VS C/C++
Types & Subtypes

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5     type All_Week is (Monday, Tuesday, Wednesday, Thursday, Friday,
6                       Saturday, Sunday);
7     subtype Week is All_Week range Monday .. Friday;
8     subtype Weekend is All_Week range Saturday .. Sunday;
9
10    Working_Day : constant Week := Sunday;
11    Free_Day : constant Weekend := Monday;
12
13
14 begin
15
16    Put_Line ("Today is");
17    Ada.Text_IO.Put_Line (Working_Day'Image);
18    Ada.Text_IO.Put_Line (Free_Day'Image);
19
20 end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:10:35: warning: value not in range of type "Week" defined at line 7
  learn.adb:10:35: warning: "Constraint_Error" will be raised at run time
  learn.adb:11:35: warning: value not in range of type "Weekend" defined at line 8
  learn.adb:11:35: warning: "Constraint_Error" will be raised at run time
$ ./learn
  raised CONSTRAINT_ERROR : learn.adb:10 range check failed
```

ADA VS C/C++
Types & Subtypes

learn.adb

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

procedure Learn is

   type All_Week is (Monday, Tuesday, Wednesday, Thursday, Friday,
                     Saturday, Sunday);
   subtype Week is All_Week range Monday .. Friday;
   subtype Weekend is All_Week range Saturday .. Sunday;

   Working_Day : constant Week := Monday;
   Free_Day : constant Weekend := Sunday;


begin

   Put_Line ("Today is");
   Ada.Text_IO.Put_Line (Working_Day'Image);
   Ada.Text_IO.Put_Line (Free_Day'Image);

end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./learn
  Today is
  MONDAY
  SUNDAY
```

ADA VS C/C++
Casting

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  tmp_1 : Integer := 100;
6  tmp_2 : Short_Integer := 24;
7
8  begin
9
10    tmp_1 := tmp_2;
11
12    Put_Line ("Values:");
13    Put_Line (tmp_1'Image);
14
15 end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:10:13: expected type "Standard.Integer"
  learn.adb:10:13: found type "Standard.Short_Integer"
  gprbuild: *** compilation phase failed
  Build failed...
```

ADA VS C/C++
Casting

learn.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  procedure Learn is
4
5  tmp_1 : Integer := 100;
6  tmp_2 : Short_Integer := 24;
7
8  begin
9
10    tmp_1 := Integer (tmp_2);
11
12    Put_Line ("Values:");
13    Put_Line (tmp_1'Image);
14
15 end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:6:01: warning: "tmp_2" is not modified, could be declared constant
$ ./learn
  Values:
  24
```

ADA VS C/C++
If & case & for

learn.adb

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

procedure Learn is

tmp_1 : Integer := 100;
tmp_2 : Short_Integer := 24;

begin

   Put_Line ("Values:");
   for tmp_1 in 1 .. 3 loop
      if Integer (tmp_2) > tmp_1 then
         tmp_2 := 2;
      elsif 30 > tmp_1 then
         tmp_2 := 10;
      else
         tmp_2 := 1;
      end if;

      Put_Line (tmp_1'Image);
      Put_Line (tmp_2'Image);
   end loop;

   Put_Line ("Values:");
   for tmp_2 in reverse 1 .. 3 loop
      Put_Line (tmp_2'Image);
   end loop;

   Put_Line ("Values:");
   case tmp_2 is
      when 1 => tmp_1 := 0;
      when 2 .. 5 => tmp_1 :=5;
      when 24 | 28 | 30 => tmp_1 :=6;
      when 34 | 36 => null;
      when others => tmp_1 := 99;
   end case;

   Put_Line (tmp_1'Image);

end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:14:16: warning: condition can only be False if invalid values present
  learn.adb:14:16: warning: condition is always True
$ ./learn
  Values:
  1
  2
  2
  10
  3
  2
  Values:
  3
  2
  1
  Values:
  5
```

ADA VS C/C++
While & exit

learn.adb

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

procedure Learn is

tmp_1 : Integer := 1;
tmp_2 : Short_Integer := 2;

begin

   Put_Line ("Values:");
   while tmp_2 < 5 loop
      if Integer (tmp_2) > tmp_1 then
         tmp_2 := 5;
      elsif 30 > tmp_1 then
         tmp_2 := 10;
      else
         tmp_2 := 1;
         exit;
      end if;

      Put_Line (tmp_1'Image);
      Put_Line (tmp_2'Image);
   end loop;

   Put_Line ("End:");

end Learn;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:5:01: warning: "tmp_1" is not modified, could be declared constant
$ ./learn
  Values:
  1
  5
  End:
```

ADA VS C/C++
Procedure & function & overloading & Interrupt handlers

learn.adb

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

type Int_t is new Integer;
type pInt_t is access Int_t;

--Deklaracja
procedure test_procedure(value : in Int_t); -- wartość
procedure test_procedure(value : in out pInt_t); -- wskaźnik
procedure test_procedure(value : in out Int_t); -- referencja

function test_function (value : in Int_t) return Int_t;


--Definicja
--Funkcja
function test_function (value : in Int_t) return Int_t is
   ret : Int_t;
begin

   return ret;
end test_function;

--Procedura
procedure test_procedure(value : in Int_t) is

begin
   Put_Line ("Value: value'Image");

end test_procedure;
```

ADA VS C/C++
Pliki - ADA

Plik nagłówkowy (plik specyfikacji) - .ads

W tym pliku umieszczamy deklaracje zmiennych, typów, podprogramów, interfejsy.

Plik źródła (plik implementacji) - .adb

W tym pliku umieszczamy ciała funkcji, procedur.

Trzeba także pamietać, że jeśli nie używamy kontenerów „package”. Stworzona procedura w pliku implementacji musi mieć tą samą nazwę jak plik.

ADA VS C/C++
Package & private

```
1   -- Interface of CrcPkg
2   package CrcPkg is
3
4      type My_array is private;
5
6      procedure Crc_Sum_Up(input_data : in  My_array;
7                           out_sum    : out Integer);
8      private
9         type My_array is array(Integer range <>) of Integer;
10  end CrcPkg;
11
12
13  -- Implementation of CrcPkg
14  package body CrcPkg is
15     procedure Crc_Sum_Up(input_data : in  My_array;
16                          out_sum    : out Integer) is
17     total : Integer;
18     begin
19        total := 0;
20        for idx in input_data'FIRST .. input_data'LAST loop
21           total := total + input_data(idx);
22        end loop;
23
24        out_sum := total;
25     end Crc_Sum_Up;
26  end CrcPkg;
27
```

ADA VS C/C++
Generic „typy/obiekty/podprogramy/wartości” - ADA

```
1   -- Interface of CrcPkg
2   generic
3      type My_array is array(Integer range <>) of Integer;
4      
5   package CrcPkg is
6      procedure Crc_Sum_Up(input_data : in  My_array;
7                           out_sum    : out Integer);
8   end CrcPkg;
9   
10  
11  -- Implementation of CrcPkg
12  package body CrcPkg is
13     procedure Crc_Sum_Up(input_data : in  My_array;
14                          out_sum    : out Integer) is
15     total : Integer;
16     begin
17        total := 0;
18        for idx in input_data'FIRST .. input_data'LAST loop
19           total := total + input_data(idx);
20        end loop;
21  
22        out_sum := total;
23     end Crc_Sum_Up;
24  end CrcPkg;
25  
```

ADA VS C/C++
Generic „typy/obiekty/podprogramy/wartości” - ADA

```
1  with Ada.Text_IO, CrcPkg;
2  use Ada.Text_IO;
3
4  procedure Learn is
5
6     ret_val_1 : Integer := 0;
7     ret_val_2 : Integer := 0;
8
9     test_array_1 : My_array(1..15);
10    test_array_2 : My_array(15..999);
11
12    package New_fun_1 is new CrcPkg(test_array_1, ret_val_1);
13    use New_fun_1;
14    package New_fun_2 is new CrcPkg(test_array_2, ret_val_1);
15    use New_fun_2;
16
17 begin
18
19    CrcPkg(Int1, Int2);         -- Uses New_fun_1.CrcPkg
20    CrcPkg(My_Int1, My_Int2); -- Uses New_fun_2.CrcPkg
21    New_fun_1.CrcPkg(Int1, Int2);
22    New_fun_2.CrcPkg(My_Int1, My_Int2);
23
24 end Learn;
25
```

ADA VS C/C++
Exceptions

```
12
13  begin
14     tmp_1 := test_function(1);
15  exception
16     when overflow => reset := 1;
17     when others => raise;
18  end;
19
20
```

- Takie błędy jak przepełnienie bufora, błędy pamięci dereferencja null jest traktowane jako wyjątek
- Każdy wyjątek może zostać obsłużony

ADA VS C/C++
Asserts

learn.adb

```
 1  with Ada.Text_IO, Ada.Assertions;
 2  use Ada.Text_IO, Ada.Assertions;
 3
 4  procedure Learn is
 5
 6     subtype Alphabet is Character range 'A' .. 'Z';
 7
 8  begin
 9     Assert('Z' = Alphabet'First, "Assert works");
10
11     Put_Line ("Learning Ada from " & Alphabet'First & " to " & Alphabet'Last);
12
13  end Learn;
```

Reset Run

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:9:15: warning: condition is always False
$ ./learn
  raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : Assert works
```

ADA VS C/C++
Design by contracts
Pre & post conditions

learn.adb

```
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3  with Ada.Numerics.Elementary_Functions;
4
5  procedure Learn is
6     pragma Assertion_Policy (Pre  => Check,
7                              Post  => Check);
8
9     function Count_area (len : Short_Integer) return Short_Integer
10       with
11          Pre => len > 0,
12          Post => Count_area'Result < 20000
13    is
14       value : Short_Integer := 0;
15    begin
16       value := len * len;
17       return value;
18    end Count_area;
19
20    tmp_1 : array(1..4) of Integer := (120, 130, 110, 140);
21
22 begin
23
24    for idx in tmp_1'Range loop
25       Put_Line ("Area: " &Count_area(Short_Integer(tmp_1(idx)))'Image);
26    end loop;
27
28 end Learn;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
  learn.adb:3:18: warning: unit "Ada.Numerics.Elementary_Functions" is not referenced
  learn.adb:20:04: warning: "tmp_1" is not modified, could be declared constant
$ ./learn
  Area: 14400
  Area: 16900
  Area: 12100
  Area: 19600
```

ADA VS C/C++
Design by contracts
Pre & post conditions

learn.adb

```
 1  with Ada.Text_IO;
 2  use Ada.Text_IO;
 3  with Ada.Numerics.Elementary_Functions;
 4
 5  procedure Learn is
 6     pragma Assertion_Policy (Pre  => Check,
 7                              Post  => Check);
 8
 9     function Count_area (len : Short_Integer) return Short_Integer
10        with
11           Pre => len > 0,
12           Post => Count_area'Result < 20000
13     is
14        value : Short_Integer := 0;
15     begin
16        value := len * len;
17        return value;
18     end Count_area;
19
20     tmp_1 : array(1..4) of Integer := (120, 130, 0, 150);
21
22  begin
23
24     for idx in tmp_1'Range loop
25        Put_Line ("Area: " &Count_area(Short_Integer(tmp_1(idx)))'Image);
26     end loop;
27
28  end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:3:18: warning: unit "Ada.Numerics.Elementary_Functions" is not referenced
  learn.adb:20:04: warning: "tmp_1" is not modified, could be declared constant
$ ./learn
  Area: 14400
  raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : failed precondition from learn.adb:11
  Area: 16900
```

ADA VS C/C++
Design by contracts
Pre & post conditions

learn.adb

```
1  with Ada.Text_IO;
2  use Ada.Text_IO;
3  with Ada.Numerics.Elementary_Functions;
4
5  procedure Learn is
6     pragma Assertion_Policy (Pre  => Check,
7                              Post  => Check);
8
9     function Count_area (len : Short_Integer) return Short_Integer
10       with
11          Pre => len > 0,
12          Post => Count_area'Result < 20000
13    is
14       value : Short_Integer := 0;
15    begin
16       value := len * len;
17       return value;
18    end Count_area;
19
20    tmp_1 : array(1..4) of Integer := (120, 130, 150, 0);
21
22 begin
23
24    for idx in tmp_1'Range loop
25       Put_Line ("Area: " &Count_area(Short_Integer(tmp_1(idx)))'Image);
26    end loop;
27
28 end Learn;
```

Console Output:

```
$ gprbuild -q -P main -gnatwa
  learn.adb:3:18: warning: unit "Ada.Numerics.Elementary_Functions" is not referenced
  learn.adb:20:04: warning: "tmp_1" is not modified, could be declared constant
$ ./learn
  Area: 14400
  raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : failed postcondition from learn.adb:12
  Area: 16900
```

ADA VS C/C++ Task/Rendezvous

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

procedure Learn is
begin

   Put_Line("This is in the main program.");

   declare
      task First_Task;
      task Second_Task;
      task Third_Task;

      task body First_Task is
      begin
         for Index in 1..4 loop
            delay 0.0;
            Put_Line("This is in First_Task.");
         end loop;
      end First_Task;

      task body Second_Task is
      begin
         for Index in 1..4 loop
            delay 0.0;
            Put_Line("This is in Second_Task.");
         end loop;
      end Second_Task;

      task body Third_Task is
      begin
         for Index in 1..8 loop
            delay 0.0;
            Put_Line("This is in Third_Task.");
         end loop;
      end Third_Task;
   begin
      delay 0.0;
      Put_Line("This is in the block body.");
      delay 0.0;
      Put_Line("This is also in the block body.");
   end; -- of declare

   Put_Line("This is at the end of the main program.");

end Learn;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./learn
  This is in the main program.
  This is in Third_Task.
  This is in Second_Task.
  This is in Third_Task.
  This is in Second_Task.
  This is in Third_Task.
  This is in Second_Task.
  This is in the block body.
  This is also in the block body.
  This is in Third_Task.
  This is in Third_Task.
  This is in First_Task.
  This is in Third_Task.
  This is in First_Task.
  This is in Third_Task.
  This is in First_Task.
  This is in First_Task.
  This is at the end of the main program.
```


Przykład zaczerpnięty ze strony https://perso.telecom-paristech.fr/

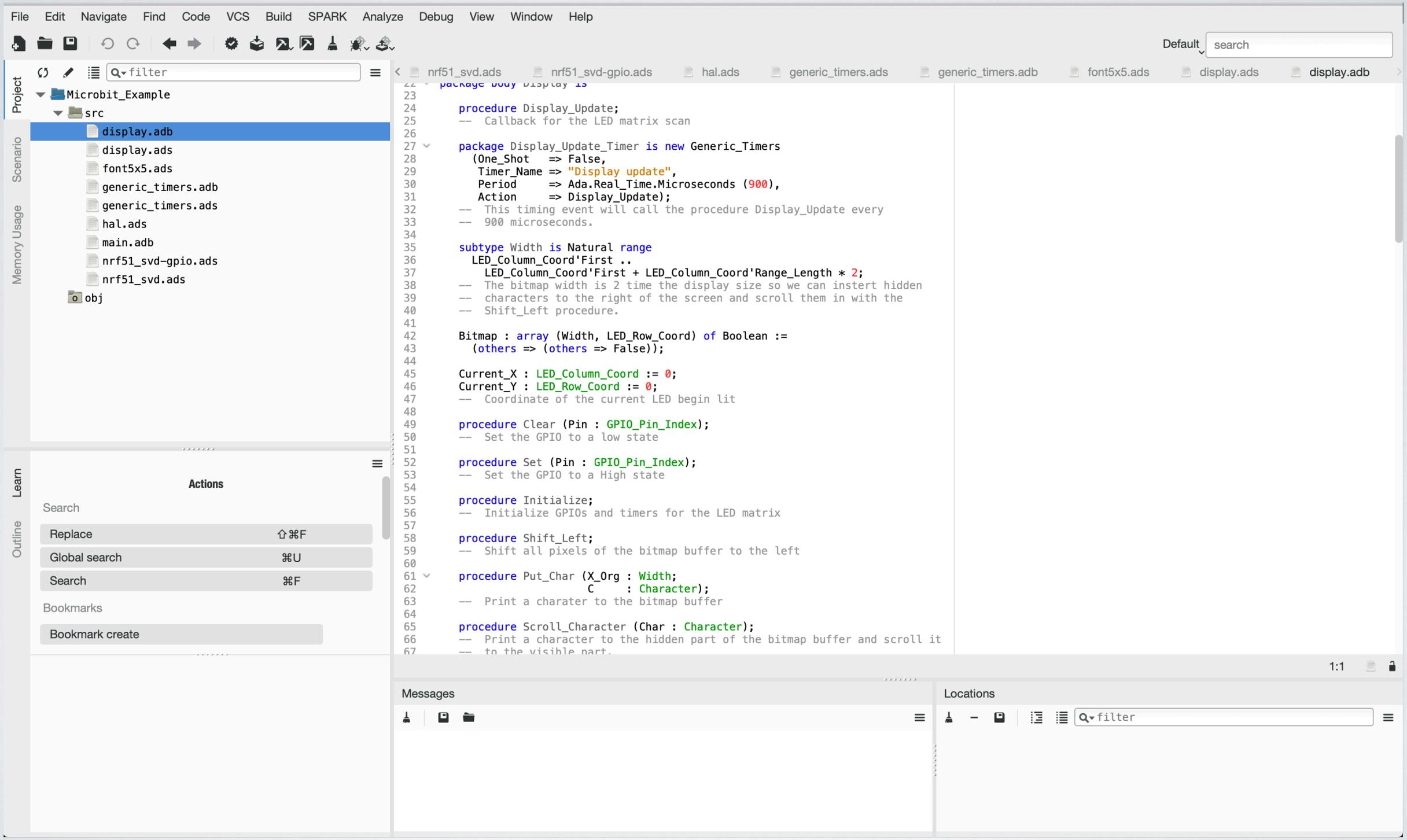
ADA VS C/C++ Task/Rendezvous

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

procedure Learn is

   task Gourmet is
      entry Make_A_Hot_Dog;
   end Gourmet;

   task body Gourmet is
   begin
      Put_Line("I am ready to make a hot dog for you");
      for Index in 1..4 loop
         accept Make_A_Hot_Dog do
            delay 0.8;
            Put("Put hot dog in bun ");
            Put_Line("and add mustard");
         end Make_A_Hot_Dog;
      end loop;
      Put_Line("I am out of hot dogs");
   end Gourmet;

begin
   for Index in 1..4 loop
      Gourmet.Make_A_Hot_Dog;
      delay 0.1;
      Put_Line("Eat the resulting hot dog");
      New_Line;
   end loop;
   Put_Line("I am not hungry any longer");
end Learn;
```

```
Console Output:
$ gprbuild -q -P main -gnatwa
$ ./learn
  I am ready to make a hot dog for you
  Put hot dog in bun and add mustard
  Eat the resulting hot dog
  Put hot dog in bun and add mustard
  Eat the resulting hot dog
  Put hot dog in bun and add mustard
  Eat the resulting hot dog
  Put hot dog in bun and add mustard
  I am out of hot dogs
  Eat the resulting hot dog
  I am not hungry any longer
```

Przykład zaczerpnięty ze strony https://perso.telecom-paristech.fr/

IDE/KOMPILATOR -GPS/GNAT



JAKIM JĘZYKIEM JEST ADA

- Na pewno niewymarłym.
- Dobrze przemyślanym.
- Umożliwiającym tworzenie projektów embedded.
- Łączącym w sobie ciekawe konstrukcje z różnych innych języków.
- Niepozwalającym na niechlujność dewelopera.
- Z ciekawym środowiskiem które ma wbudowane narzędzia do tworzenia unit testów i analizy kodu.