ADA USER JOURNAL

Volume 41
Number 4
December 2020

Contents

Quarterly News Digest

Alejandro R. Mosteo

Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain; email: amosteo@unizar.es

Contents

[Messages without subject/newsgroups are replies from the same thread. Messages may have been edited for minor proofreading fixes. Quotations are trimmed where deemed too broad. Sender's signatures are omitted as a general rule. —arm]

Preface by the News Editor

Dear Reader,

As I write these lines I have the FOSDEM livestream on my second monitor. This brings me to the first topic that I want to highlight in this issue: sadly, during last quarter we knew [1] of the passing of fellow Adaist Vinzent "Jellix" Höfler. I "devirtualized" him precisely at FOSDEM'20, where he cracked a joke during my demo that was producing lots of "No C sources found in this project" warnings. To this, he had to say (filtered by my memory): "I don't see the problem."

As for regular discussions, this time around I selected a few interesting and sometimes amusing heated debates. We have a couple of technical rabbit holes, about the finer details of protected actions syntax (that started from an innocent-looking question about logging [2]) and properties of real-time clocks and durations [3]. Did you know that Duration'Range can legally be as short as a day? I am a bit ashamed to admit I did not. Also, an often-seen observation about array indexing syntax from an Ada newcomer led to many strongly-held opinions on the merits (or lack thereof) of some aspects of Ada syntax [4] that led us as far as when Ada prototypes had parentheses for subprograms without arguments.

To conclude, during this period also took place the Advent of Code, a scored competition in which a programming puzzle a day is presented for you to solve in your favorite language. A few members of c.l.a. took the bait and this led to some interesting exchanges of ideas around the solutions in a large number of threads which I have strived to summarize for you [5].

Sincerely,
Alejandro R. Mosteo.

[1] "Tragic News about Vinzent Hoefler", in Obituary.

[2] "Logging and Protected Actions", in Ada Practice.

[3] "Starting time of Real-time Clock", in Ada Practice.

[4] "Ada Syntax Questions", in Ada Practice.

[5] "Advent of Code" and "Advent of Code Thread Compilation", in Ada Practice.

Ada-related Events

ACM HILT 2020 at SPLASH 2020

[Event already in the past, for the record. —arm]

From: Richard Wai <ric.wai88@gmail.com>
Subject: ACM HILT 2020 (High Integrity Language Technologies) at SPLASH 2020 - Nov 16 & 17
Date: Sun, 1 Nov 2020 19:56:24 -0800
Newsgroups: comp.lang.ada

Hey everyone, just a reminder that the 6th HILT workshop this year is on Nov 16 & 17, and is part of the larger SPLASH 2020 conference (2020.spashcon.org). Unsurprisingly, this year's workshop will be fully virtual.

HILT 2020 focuses on the growing importance of large-scale, highly parallel, distributed and/or cloud applications.

For Ada specifically, we have talks on:

- A layered mapping of Ada 202X parallel constructs to OpenMP (Tucker Taft),
- Experience integrating FAA's NextGen ERAM (mostly Ada) with SWIM (Mixed languages) (Brian Kleinke, Leidos)
- A highly parallel multiple double precision polynomial solver framework in Ada (PHC Pack - Prof. Jan Verschelde of UoI at Chicago)
- A cloud-native/HPC-centric hyperscaling framework for Ada, and a supporting Ada-specific exokernel OS (Yours truly)

Please check out the workshop's website (https://2020.splashcon.org/home/hilt-2020) if you are interested in attending.

CfC 25th Ada-Europe Conf. on Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 25th Ada-Europe Conf. on Reliable Software Technologies
Date: Sun, 6 Dec 2020 11:39:55 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

[CfC is included in the Forthcoming Events Section —arm]

Ada-Europe 2021 Conference - Extended 14 January Deadline

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada-Europe 2021 Conference - EXTENDED 14 January deadline
Date: Thu, 31 Dec 2020 15:54:46 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

The Ada-Europe 2021 Conference organizers decided to provide more time for authors to prepare their contributions. The deadline for most submissions is extended to Thursday 14 January 2020. 2 weeks remain!

[CfC is included in the Forthcoming Events Section —arm]

Happy Birthday, Lady Ada

From: AdaMagica <christ-usch.grein@t-online.de>
Subject: Happy birthday, Lady Ada
Date: Wed, 9 Dec 2020 19:00:53 -0800
Newsgroups: comp.lang.ada

Primeval times when Babbage dwelt:

not were bit nor byte

nor operating system,

not hardware below

nor above software,

abyss abundant,

but computer nowhere.

And lo, there was Ada,

and Ada separated the numbers

and split them,

in Zero and One did she split them.

Continuation see: https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada%20Magica/20.html

From: Simon Wright <simon@pushface.org>
Date: Thu, 10 Dec 2020 10:08:56 +0000

> in Zero and One did she split them.

The Analytical Engine was a decimal machine

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Thu, 10 Dec 2020 02:52:06 -0800

> The Analytical Engine was a decimal machine

That's OK.

I know Babbage's engine came before Zuse, C++ came after Ada.

But an ode need not be historically correct. Would you claim Edda is historically correct?

Ár var alda, þat er Ymir bygði,
Vara sandr né sær né svalar unnir;
iorð fannz æva né upphiminn,
gap var ginnunga, enn gras hvergi.

Translate this and it will give about the same as the first verse above.

Ada and Education

Strategies for Teaching Ada

[Cont. from "Strategies for Teaching Ada" in AUJ 41-2, June 2020 —arm]

From: Norman Worth <nworth@comcastnospam.net>
Subject: Re: Beginning Ada Programming, by Andrew T. Shvets (2020)
Date: Mon, 2 Nov 2020 14:14:03 -0700
Newsgroups: comp.lang.ada

>> There's nothing wrong with using integer to start off and then moving onto defined types.

> Yes there is! (see my paper at the last Ada-Europe). The first message when you teach Ada is that it is all about defining proper types. You have to start by fighting bad habits from other languages.

One of the most difficult things for programmers to graft these days is the concept and proper use of types, which is key to Ada. Ada makes this even more complicated with the very useful attributes of private and limited types. Unless a text clearly conveys the use of types and illustrates it throughout, it is useless for teaching people Ada. Since this is a foreign concept to most current programmers, illustrations and good exercises are needed, too.

Compare this text to Barnes, which most of us use as a quick reference.

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 12 Nov 2020 13:24:57 -0800

> So I can't learn Ada from docs online?

You can. But the best Ada resources are books and the Language Reference.

(The Language Reference is dry, but very readable compared to some of the other standards I've come across.)

Also, the compiler itself is typically very good because of generally high-quality error messages.

From: Chris Townley <news@cct-net.co.uk>
Date: Thu, 12 Nov 2020 22:31:59 +0000

> Also, the compiler itself is typically very good because of generally high-quality error messages.

Although the errors can be very confusing sometimes, if you make a big mistake…

Ada-related Resources

[Delta counts are from Nov 2nd to Feb 2nd. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Tue, 02 Feb 2021 17:31:21 +0100
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_078 (+53) members [1]
- Reddit: 5_233 (+513) members [2]
- Stack Overflow:1_973 (+49) questions [3]
- Freenode: 85 (-5) users [4]
- Gitter: 66 (+2) people [5]
- Telegram: 108 (+18) users [6]
- Twitter: 60 (-7) tweeters [7]
  95 (+3) unique tweets [7]

[1] https://www.linkedin.com/groups/114211/

[2] http://www.reddit.com/r/ada/

[3] http://stackoverflow.com/questions/tagged/ada

[4] https://netsplit.de/channels/details.php?room=%23ada&net=freenode

[5] https://gitter.im/ada-lang

[6] https://t.me/ada_lang

[7] http://bit.ly/adalang-twitter

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Mon, 02 Nov 2020 18:41:21 +0100
To: Ada User Journal readership

Rosetta Code: 761 (+14) examples [1]
37 (=) developers [2]

GitHub: 755 (+26) developers [3]

Sourceforge: 278 (+2) projects [4]

Open Hub: 212 (=) projects [5]

Alire: 146 (+16) crates [6]

Bitbucket: 88 (-2) repositories [7]

Codelabs: 52 (=) repositories [8]

AdaForge: 8 (=) repositories [9]

[1] http://rosettacode.org/wiki/Category:Ada

[2] http://rosettacode.org/wiki/Category:Ada_User

[3] https://github.com/search?q=language%3AAda&type=Users

[4] https://sourceforge.net/directory/language:ada/

[5] https://www.openhub.net/tags?names=ada

[6] https://alire.ada.dev/crates.html

[7] https://bitbucket.org/repo/all?name=ada&language=ada

[8] https://git.codelabs.ch/?a=project_index

[9] http://forge.ada-ru.org/adaforge

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Mon, 20 Jul 2020 09:38:21 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. The IEEE ranking has published its 2020 edition. —arm]

- TIOBE Index: 32 (+7) 0.4% (+0.05%) [1]
- PYPL Index: 19 (=) 0.65% (+0.3%)[2]
- IEEE Spectrum (general): 39 43 (+4) Score: 32.8 24.8 [3]
- IEEE Spectrum (embedded): 12 13 (+1) Score: 32.8 24.8 [3]

[1] https://www.tiobe.com/tiobe-index/

[2] http://pypl.github.io/PYPL.html

[3] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020

Ada-related Tools

Zip-Ada v.57

From: gautier_niouzes@hotmail.com
Subject: Ann: Zip-Ada v.57
Date: Fri, 2 Oct 2020 09:57:42 -0700
Newsgroups: comp.lang.ada

New in v.57 [rev. 799]:

- UnZip: fixed bad decoding case for the Shrink (LZW) format, on some data compressed only by PKZIP up to v.1.10, release date 1990-03-15.
- Zip.Create: added Zip_Entry_Stream_Type for doing output streaming into Zip archives.
- Zip.Compress: Preselection method detects Audacity files (.aup, .au) and compresses them better.

Zip-Ada is a pure Ada library for dealing with the Zip compressed archive file format. It supplies:

- compression with the following sub-formats ("methods"): Store, Reduce, Shrink (LZW), Deflate and LZMA
- decompression for the following sub-formats ("methods"): Store, Reduce, Shrink (LZW), Implode, Deflate, Deflate64, BZip2 and LZMA
- encryption and decryption (portable Zip 2.0 encryption scheme)
- unconditional portability - within limits of compiler's provided integer types and target architecture capacity
- input archive to decompress can be any kind of indexed data stream
- output archive to build can be any kind of indexed data stream
- input data to compress can be any kind of data stream
- output data to extract can be any kind of data stream
- cross format compatibility with the most various tools and file formats based on the Zip format: 7-zip, Info-Zip's Zip, WinZip, PKZip, Java's JARs, OpenDocument files, MS Office 2007+, Google Chrome extensions, Mozilla extensions, E-Pub documents and many others
- task safety: this library can be used ad libitum in parallel processing
- endian-neutral I/O

Main site & contact info:

http://unzip-ada.sf.net

Project site & subversion repository:

https://sf.net/projects/unzip-ada/

GitHub clone with git repository:

https://github.com/zertovitch/zip-ada

GNAT CE 2020, arm-eabi, for macOS

From: Simon Wright <simon@pushface.org>
Subject: GNAT CE 2020, arm-eabi, for macOS
Date: Tue, 06 Oct 2020 16:59:05 +0100
Newsgroups: comp.lang.ada

There were few downloads of this from the AdaCore site, so they've not produced a 2020 edition. This is my attempt at that missing edition!

At https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-arm-eabi-darwin-bin/

Simple Components v4.51

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada v4.51
Date: Sun, 18 Oct 2020 08:43:41 +0200
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, directed graphs, sets, maps, B-trees, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, multiple connections server/client designing tools and protocols implementations.

http://www.dmitry-kazakov.de/ada/components.htm

Changes to the previous version:

- Memory leak fixed in the package Generic_Unbounded_Ptr_Array;
- Bug fix in data selector initialization (in the package GNAT.Sockets.Connection_State_Machine);
- An exception-free variant of Put was added to the Generic_Indefinite_FIFO package;
- ModBus TCP/IP implementation bug fix (the package GNAT.Sockets.Connection_State_Machine.MODBUS_Client);
- Code modified to work around GCC 10.0.1 optimization bug.

Simple Components v4.53

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple components v4.53
Date: Sun, 13 Dec 2020 10:02:03 +0100
Newsgroups: comp.lang.ada

[...]

Changes to the previous version:

- Get_Reader_Timeout, Set_Reader_Timeout, Wait_For_Tasks were added to the package GNAT.Sockets.Server.Blocking;
- JSON parser fixed to accept empty objects {} and empty array [];
- OpenSSL bindings were extended;
- The procedure Next_Unbiased was added to the package Generic_Random_Sequence.

Ahven 2.8

From: Tero Koskinen <tero.koskinen@iki.fi>
Subject: ANN: Ahven 2.8
Date: Sun, 18 Oct 2020 21:47:38 +0300
Newsgroups: comp.lang.ada

Ahven version 2.8 is now available from https://www.ahven-framework.com/

Direct links to tar.gz and zip packages:

* https://www.ahven-framework.com/releases/ahven-2.8.tar.gz
* https://www.ahven-framework.com/releases/ahven-2.8.zip

Ahven is a simple unit test library (or a framework) for Ada programming language. It is loosely modelled after JUnit and some ideas are taken from AUnit. Ahven is free software distributed under permissive ISC license and should work with any Ada 95, 2005, or 2012 compiler.

Version 2.8 is a minor maintenance release. The changes are:

* Source code repository of Ahven is now hosted at

  Sourcehut: https://hg.sr.ht/~tkoskine/ahven
* Improvements to Janus/Ada build scripts
* Improvements to GNAT build scripts
* Minor documentation updates

HAC v.0.076

From: gautier_niouzes@hotmail.com
Subject: Ann: HAC v.0.076
Date: Sat, 24 Oct 2020 00:38:57 -0700
Newsgroups: comp.lang.ada

HAC (HAC Ada Compiler) is a small, quick, open-source Ada compiler, covering a subset of the Ada language.

You find below the changes since the last post about HAC in this forum.

Links to the project and contact (tracing ;-)) addresses are available from the blog posts cited below.

0.071 Discrete type range is stored in type definition; "subtype T1 is T2;"

0.072 Subtype_Indication (e.g. "for B in Boolean loop", "array (States) of Prob")

https://gautiersblog.blogspot.com/2020/06/hac-v0072-subtype-indication.html

0.073 The VM can be aborted via the Feedback procedure

0.074 Types: Time and Duration

0.075 Added Ada.Calendar-like functions

https://gautiersblog.blogspot.com/2020/10/hac-v0075-time-functions-goodies-for.html

0.076 Added Ada.Directories-like subprograms

https://gautiersblog.blogspot.com/2020/10/hac-v0076-adadirectories-like.html

XNAdaLib 2020 Binaries for macOS Catalina

From: Blady <p.p11@orange.fr>
Subject: [ANN] XNAdaLib 2020 binaries for macOS Catalina including GTKAda and more.
Date: Sun, 25 Oct 2020 10:11:49 +0100
Newsgroups: comp.lang.ada

This is XNAdaLib 2020 built on macOS 10.15 Catalina for Native Quartz with GNAT Community 2020 including:

- GTKAda 20.2 (www.adacore.com/gtkada) with GTK+ 3.24.20 (www.gtk.org) complete,
- Glade 3.22.1 (glade.gnome.org),
- GnatColl 20.2 (github.com/AdaCore/gnatcoll),
- Florist mid-2020a (github.com/Blady-Com/florist),
- AdaCurses 6.2 (invisible-island.net/ncurses/ncurses-Ada95.html),
- Gate3 0.5c (sourceforge.net/projects/lorenz),
- Components 4.50 (www.dmitry-kazakov.de/ada/components.htm),
- AICWL 3.24 (www.dmitry-kazakov.de/ada/aicwl.htm),
- Zanyblue 1.4.0 (zanyblue.sourceforge.net),
- PragmARC mid-2020 (pragmada.x10hosting.com/pragmarc.htm),
- GNOGA 1.6-beta (www.gnoga.com),
- SparForte 2.3.1 (sparforte.com),
- Alire 0.6.1 (alire.ada.dev), NEW

and as side libraries:

- Template Parser 20.2,
- gtksourceview 3.24.4,
- GNUTLS 3.6.14,
- GMP 6.1.2,
- make 4.2.1,
- Python 2.7.17.

XNAdaLib binaries have been post on Source Forge:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-catalina

Report preferably all comments to MacAda.org mailing list:

http://macada.org/macada/Contacts.html

See list archive:

https://hermes.gwu.edu/archives/gnat-osx.html

From: Simon Wright <simon@pushface.org>
Date: Sun, 25 Oct 2020 09:39:41 +0000

Great stuff, just a couple of comments -

> - Python 2.7.17.

Not maintained since 1 Jan. There are excellent downloads of 3 (currently 3.9) at python.org.

> Report preferably all comments to MacAda.org mailing list:

> http://macada.org/macada/Contacts.html

Gives a (Korean?) 404.

You can subscribe at https://hermes.gwu.edu/cgi-bin/wa?A0=GNAT-OSX

RFC UXStrings Package.

From: Blady <p.p11@orange.fr>
Subject: RFC UXStrings package.
Date: Wed, 11 Nov 2020 21:18:17 +0100
Newsgroups: comp.lang.ada

UXStrings is now a standalone library available on Github.

https://github.com/Blady-Com/UXStrings

Comments on specifications are welcome.

A first implementation POC is provided. UTF-8 encoding is chosen for internal representation. The Strings_Edit library is used for UTF-8 encoding management.

http://www.dmitry-kazakov.de/ada/strings_edit.htm

This implementation which is only to demonstrate the possible usages of UXString has many limitations as for instance there is no memory deallocation. Only a few API are implemented.

A test program is also provided with some features working.

See readme for full details.

https://github.com/Blady-Com/UXStrings/blob/master/readme.md

From: Vadim Godunko <vgodunko@gmail.com>
Date: Fri, 27 Nov 2020 00:38:56 -0800

There are few more options to forget about encodings and related issues:

New AdaCore's VSS
https://github.com/AdaCore/VSS

Old Matreshka's League
http://forge.ada-ru.org/matreshka

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 27 Nov 2020 11:05:08 +0000
> There are few more options to forget about encodings and related issues:

My very basic utf-8 string - https://github.com/Lucretia/uca

Ada-12 Version of PragmARC

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Ann] Ada-12 Version of the PragmAda Reusable Components
Date: Sun, 1 Nov 2020 19:20:42 +0100
Newsgroups: comp.lang.ada

Now that there are 2 (count 'em!) Ada-12 compilers*, an Ada-12 version of the PragmARCs is available at https://github.com/jrcarter/PragmARC

In addition to making use of Ada-12 features, this version has a restructured package hierarchy and is released under the 3-clause BSD license.

These have only been compiled with the GNAT compiler. Feedback from those with access to the other compiler would be welcome.

*Defined as a compiler that implements the entire Ada-12 core language.

SweetAda 0.1g

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda 0.1g released
Date: Sun, 15 Nov 2020 13:16:55 -0800
Newsgroups: comp.lang.ada

I've just released SweetAda 0.1g.

This is a maintenance release, and introduces new toolchains based on Binutils 2.35, GCC 10.2.0 and GDB 10.1.

Along with new tools, the basic support libraries, e.g., GMP, MPFR, MPC, and all auxiliary libraries were used at the highest stable version during the builds.

Sorry for a significant delay in releasing, but it is very time-consuming to keep everything in-sync, especially when toolchains change. Neither I had the time to complete the manual, I'll try to do that in the near future.

SweetAda itself gets few changes:

- due to a deeper Ada code analysis, the new compiler front-end showed possible superfluous aspects; they are removed and warnings made silent
- slightly better menu scripts
- echo_log() and echo_log_error() functions in Bash scripts are now renamed as log_print() and log_print_error()
- minor changes and typos here and there

Of course, LibGCC and RTS packages are synchronized with new toolchains, so download them as well.

I am working on Insight too, hopefully packages will be available ASAP, but it is still at 20200417 timestamp. Please note that if you install Insight, it will overwrite the standard GDB executable, and you're stuck at 9.1. GPRbuild remains at 20200417 timestamp as well.

I discovered a mismatch in QEMU for Linux 20200817 targeted for ARM, AVR, AArch64, x86 and M68k CPUs, where executables end up being objects for an OS X platform, because of bad naming. This is now corrected. Sorry for that, please re-download the following packages:

qemu-aarch64-20200817L.tar.xz

qemu-arm-20200817L.tar.xz

qemu-avr-20200817L.tar.xz

qemu-i386-20200817L.tar.xz

qemu-m68k-20200817L.tar.xz

Furthermore, QEMU for Windows packages lack libffi-6.dll. This is now corrected. Please re-download

qemu-<every_cpu>-20200817W.zip (or place a libffi-6.dll library taken from a random MinGW64 package, along the QEMU executable).

Find everything at https://www.sweetada.org.

By the way, the connection to SweetAda website is now completely secure. Many thanks to the Certbot team.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Mon, 16 Nov 2020 12:51:39 -0800

I suggest that an announcement like this should include, at or near the top of the article, a brief description of what SweetAda is.

From the web site:

SweetAda is a lightweight development framework whose purpose is the implementation of Ada-based software systems.

[...]

AdaStudio-2021 Release 01/01/2021 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce : AdaStudio-2021 release 01/01/2021 free edition
Date: Wed, 30 Dec 2020 00:51:09 -0800
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2021.

In the new AdaStudio release it was added Qt6Ada support for new framework Qt-6.0.0.

I added some packages from Qt-5.15.0 open source (qtcharts qtconnectivity qtgraphicaleffects qtimageformats qttexttospeech qtlocation qtlottie qtmultimedia qtsensors qtserialbus qtserialport qtwebchannel)

Qt6ada version 6.0.0 open source and qt6base.dll, qt6ext.dll (win64), libqt6base.so, libqt6txt.so(x86-64) built with Microsoft Visual Studio 2019 x64bin Windows, gcc x86-64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux x86-64 Debian 10.0

I built Qt6 binaries for win64 and x86-64 and include them into AdaStudio-2021 (qt6ada directory)

Known problems:

1) for quick3d and quickcontrols2 plugins I have got unresolved entry points ml_registr_types_QtQuick3D(), so some examples do not work properly.
2) in Linux multimedia plugins do not built properly and services do not work (qtavada works fine)
3) webengine does not work and it is not added to qt6ada

Qt 6 is a new long time project and I hope to solve these problems in the next release.

Qt6Ada is built under a GNU GPLv3 license: https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada and VTKAda for Windows, Linux (Unix) is available from

https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio

web page or Google drive

https://drive.google.com/folderview?id=0B2QuZLoe-yiPbmNQRl83M1dTRVE&usp=sharing (google drive. It can be mounted as virtual drive or directory or viewed with Web Browser)

The full list of released classes is in "Qt6 classes to Qt6Ada packages relation table.docx"

The latest hacker attacks will force many companies to reconsider technologies based on scripting languages such as Python, Ruby, Perl, JavaScript and others, in which it is much easier to replace code than in translated modules. Therefore, interest in a language such as Ada should greatly increase.

If you have any problems or questions, let me know.

Ada-related Products

Adalog's "Back to Quality" Program

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] Adalog's "Back to Quality" program
Date: Sat, 24 Oct 2020 09:03:33 +0200
Newsgroups: comp.lang.ada

Adalog annonces the "Back to quality" program.

Thanks to our experience and advanced tools, we offer technical assistance to relieve your technical dept by fixing non-conformities to your coding standard that you never have time to fix by yourself.

For more details, see:
https://adalog.fr/en/btq_program.html
or write to info@adalog.fr

State Preserving Fault Tolerance for Ada Applications

From: Thomas Wetmore <tom.wetmore@gmail.com>
Subject: State Preserving Fault Tolerance for Ada Applications
Date: Wed, 9 Dec 2020 13:37:51 -0800
Newsgroups: comp.lang.ada

Our small startup has developed a new software fault tolerant (FT) architecture, implemented as an SDK and library, that we are currently adapting for use with Ada and SPARK. It will enable developers to create true state preserving, fault tolerant Ada applications by either developing new or modifying existing code. The architecture provides additional levels of availability and security by providing resilience against both

hardware failures and software anomalies (attacks). The port will enable Ada users to create FT Ada applications that can be adapted for most COTS h/w - s/w platforms. Such applications can even be run on heterogeneous, geographically distributed configurations - using bare metal, virtual machines, or containers.

Note that this new application-based FT software technology was created by our veteran computer design engineers who have developed multiple generations of fault tolerant systems currently in world-wide use. The Ada implementation of the technology is being created by a veteran Ada expert who has been developing with Ada since its inception.

We are looking for users with whom we can collaborate to 1) provide needs input, 2) assist with QC & real-world use case testing, and/or 3) create prototypes and/or proofs of concept. Please let me know if you are interested in learning more and we will be glad to share additional information.

Ada and Operating Systems

Developing on a Mac

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: Developing on a Mac
Date: Wed, 14 Oct 2020 09:39:58 -0700
Newsgroups: comp.lang.ada

I searched but could not find it. How to develop Ada programs on a Mac today (Catalina)? GNAT CE 2020 for Mac has no GPS anymore. Must one use Xcode? How to make Xcode Ada-aware and integrate it with GNAT? Some other Ada-aware IDE for Mac?

From: Simon Wright <simon@pushface.org>
Date: Wed, 14 Oct 2020 20:02:59 +0100

> How to develop Ada programs on a Mac today (Catalina)? GNAT CE 2020 for Mac has no GPS anymore.

If you want GPS the best bet is probably to use the GPS from GNAT CE 2019 with the new compiler. Have CE 2020 bin first on your PATH, then explicitly call up gps: I just used /opt/gnat-ce-2019/bin/gps.

There is a port of GNAT Studio to Catalina[1], but ISTR it's not all working 100%?

> Must one use Xcode? How to make Xcode Ada-aware and integrate it with GNAT?

Last time I heard, Xcode is proprietary and closed, and no one has ever reported extending it for Ada. But of course I haven't been looking.

> Some other Ada-aware IDE for Mac?

Emacs[2], with ada-mode[3]].

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2020-catalina/GNATStudio-20.2-a.dmg/download

[2] https://emacsformacosx.com

[3] https://www.nongnu.org/ada-mode/ada-mode.html

From: Simon Wright <simon@pushface.org>
Date: Thu, 15 Oct 2020 10:35:39 +0100

> How to develop Ada programs on a Mac today (Catalina)?

[...]

> Some other Ada-aware IDE for Mac?

Just announced:

https://github.com/thindil/vim-ada/releases/tag/v10.0

https://github.com/thindil/Ada-Bundle

From: Jerry <list_email@icloud.com>
Date: Thu, 15 Oct 2020 16:41:40 -0700

> How to develop Ada programs on a Mac today (Catalina)?

Some of the following is kind of vague but I hope it is useful. Many listers will know much more.

One time a long time ago someone (on this list?) made Xcode work with Ada. It was fantastic. Even a debugger IIRC. But apparently Apple likes to change the underpinnings and after some time Xcode ceased to work with Ada. (There also is or was a FPC Pascal way with Xcode that was even more capable but I haven't checked into that for a long time.)

There also used to be Carbon bindings to Ada, possibly made by the same person. (The words "Blady" and "Pascal" come to mind for this person.) They were on the macada.org web site which doesn't seem to do much these days, as well as being linked from AdaPower. Of course the Carbon API has been long-deprecated but I'm sure it is still used. (How does Microsoft keep Word et al working on Macs?)

It's not a full IDE in some opinions but Visual Studio Code runs on Macs, even my now-ancient 2008 PowerBook and macOS 10.11.6. There is an Ada plug-in but make sure you get the right one. I think this plug-in might be supported by AdaCore. And there's something about an Ada Language Server. I don't really understand all of this. I've tried to get this running but the instructions are minimal so it is taking more effort than it should. (Why are installation instructions so frequently written assuming that you already know how to install stuff?)

IntelliJ IDEA CE also has an Ada plug-in. I guess Eclipse has an Ada plug-in as well. I think AdaCore supports this but I'm not sure if the Mac version is well-supported.

None of the above except Xcode is a native Mac app so you'll have to deal with a certain amount of cross-platform-turdism. I would happily pay hundreds of $US for a native Mac Ada IDE but that will never happen. The previously-mentioned Xcode hack was close enough, though.

There are lots of text editors that aren't too bad. I have used Textmate with its Ada plug-in (bundle) which I've modified for my own purposes for many years. Not an IDE but it does have the capability to link from parsed error reports back to your code. Textmate was a leader in this area and its bundle architecture has been used by several other editors.

Sorry if this is all a little sketchy.

Now for something OT. If you are doing technical work in Ada and want to store or examine or plot results, I have made Igor Pro (wavemetrics.com) work with Ada. This is a fantastic arrangement. It's almost as nimble as working in a notebook (think Jupyter or Jupyter Lab) but you get the awesomeness of Igor Pro to plot, post-process, and document.

From: Blady <p.p11@orange.fr>
Date: Fri, 16 Oct 2020 22:21:30 +0200

> There also used to be Carbon bindings to Ada, possibly made by the same person.

If I remember well, the Carbon bindings were provided by James E. Hopper from a Pascal to Ada translation with p2ada of Apple Carbon API in Pascal. Though Carbon may still work, Apple wasn't maintaining the Pascal API, but only the C API.

Thus Ada Carbon Bindings weren't used anymore as far as I know. I provided some Xcode support to Ada but after, as you said, Xcode was no more customizable.

You'll find here some historical material:

https://blady.pagesperso-orange.fr/alpha.html

Ada on QNX

From: DrPi <314@drpi.fr>
Subject: Ada on QNX
Date: Thu, 10 Dec 2020 08:50:53 +0100
Newsgroups: comp.lang.ada

Anyone has cross-compiled Ada for QNX SDP 6.6.0 (ARM target)?

From: Quentin Ochem <qochem@gmail.com>
Date: Thu, 10 Dec 2020 08:03:48 -0800

Hi Nicolas,

FWIW, there's an AdaCore port that has been done specifically targeting QNX/ARM. If you want to discuss, feel free to drop me an e-mail (ochem@adacore.com).

From: DrPi <314@drpi.fr>
Date: Fri, 11 Dec 2020 10:49:57 +0100

Yes, I know. I've been in contact with someone from Adacore about 2 years ago. But the port is for QNX SDP 7.0.0 and later only.

It seems that there is provision for a QNX compilation in FSF GNAT. Not sure of that and not tried to go this way yet.

Read/Write Access to UNIX Character Devices

From: philip.munts@gmail.com
Subject: Read/write access to Unix character devices
Date: Sun, 20 Dec 2020 20:59:28 -0800
Newsgroups: comp.lang.ada

Lately I have been working with Unix (really Linux, FreeBSD, and OpenBSD) character devices (these happen to be USB raw HID devices, but the problem is more general than that). The way these work is that each hardware device has a character device node file in /dev/, like /dev/hidraw1. You open the file for both read and write access. Then you can send a command to the device by writing a binary blob and get a response by subsequently reading a binary blob. For what I am doing, it is important not to block on reads forever if there is no response forthcoming, so I need at least read timeouts.

So far, I have been binding the C library functions open(), close(), read(), write(), and poll() with pragma Import. That works, but I have wondered if there is some way of accomplishing the same thing more portably. The packages GNAT.Sockets and GNAT.Serial_Communicatons can be viewed as special case solutions, but I would like a general solution.

What I would really like is Ada.Sequential_IO with InOut_File and a timeout mechanism, perhaps like the select() wrapper in GNAT.Sockets.

So far I haven't found anything in the Ada. or GNAT. that supports InOut_File semantics (other than Direct_IO) let alone timeouts. Does anybody have any suggestions?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:23:08 -0600

I would use Stream_IO for this, but you'd need help from your implementer to get timeouts/nonblocking I/O. If they have them, they'd be some sort of Form parameter (that's what the typically ignored Form parameter is for).

Stream_IO is a lot more flexible that Sequential_IO and Direct_IO. (Some implementations implement those older Ada 83 packages in terms of Stream_IO.)

Ada and Other Languages

Importing Python Library into Ada

From: Roger Mc <rogermcm2@gmail.com>
Subject: Import Python library into an Ada package?
Date: Thu, 3 Dec 2020 23:36:13 -0800
Newsgroups: comp.lang.ada

Is it possible to import a Python library, such as graphviz, into an Ada package? So far I have only been able to find information on exporting Ada to Python.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 4 Dec 2020 11:23:21 +0100

I am not sure what you mean. Python is not a compiled language, so formally speaking a Python library is not a library and you cannot import it in the sense of linking it to your application and calling subprograms from it using certain calling conventions.

If you rather meant whether you could execute a Python script from Ada while passing parameters to it and taking results from, yes you can. If that script were a part of some Python module, yes you can load it and once loaded call (interpret) functions from the module.

P.S. Before you proceed, Python is a huge mess and interfacing it is a pain in the ... So you should consider if Graphviz is worth the effort. If you find a GTK or Qt library that is doing approximately the same, that would be a wiser choice, IMO.

From: Roger Mc <rogermcm2@gmail.com>
Date: Fri, 4 Dec 2020 03:37:53 -0800

Many thanks for your prompt response and comments Dmitry; they are well appreciated with some of the contents somewhat expected.

I think that I misused the term " Python library"; I think "Python module" is what I should have used.

In this context, in Python, is a module a script? I'll investigate this.

[...]

The project that I am embarking on is to use Ada for an on-line course in machine learning that uses Python as its teaching platform. The importing that I was contemplating concerns special machine learning Python modules used in the course.

Of course, the alternative is for me to translate the Python modules into Ada which is something I've done in the past; generally, in my opinion, yielding much better and more readable code. Again, thanks for your very helpful comments which, hopefully, have focused my mind on the way ahead.

Regarding your comment that "Python is a huge mess" and my own opinion of Python; I am mortified that Python seems to have become the standard language for teaching computer programming and, particularly, that it seems to be the choice of leading university computer science courses. It seems that the old well-established rules of quality computer program design have been completely abandoned by these institutions.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 4 Dec 2020 15:22:46 +0200

> Is it possible to import a Python library, such as graphviz, into an Ada package?

If you mean the Graphviz tool-set, https://en.wikipedia.org/wiki/Graphviz, that seems to be written in C and to be open source. You should be able to call Graphviz functions from Ada in the same way as one calls any C code from Ada. The Python module you refer to is probably just a binding from Python to the C code in Graphviz.

If you want to use Graphviz just to draw automatically laid-out graphs, there is another way, that I have used: make the Ada program write out the graph definition as a text file in the "dot" language, and then invoke the "dot" program from Graphviz to lay out and draw the graph into some graphical format. However, it may be troublesome to make this method work interactively -- I was satisfied with non-interactive post-processing of the "dot" file generated by my Ada program.

From: gautier_niouzes@hotmail.com
Date: Fri, 4 Dec 2020 05:41:08 -0800

As a side note, there is a cool utility called DePlo (https://sites.google.com/site/depplot/ , sources here : https://launchpad.net/deplo) that creates a dependency graph of Ada units from the .ali files that GNAT produces when building a project.

This graph is in Graphviz's DOT format.

And indeed, graphviz is not specific to Python. The sources are in C, and the Web site mentions bindings to: guile, perl, python, ruby, C#, tcl .

From: Simon Wright <simon@pushface.org>
Date: Fri, 04 Dec 2020 13:55:15 +0000

> Regarding your comment that "Python is a huge mess" and my own opinion of Python; [...]

I'd certainly agree that interfacing to Python from Ada is a huge mess (specifically, unsupported hand management of garbage collection, as you have to do if invoking Python objects).

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 4 Dec 2020 19:32:58 +0100

> If you want to use Graphviz just to draw automatically laid-out graphs [...]

And if you really just want to draw graphs - and can use another tool - gnuplot can be controlled by spawning it and sending commands on stdin via pipes.

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Fri, 4 Dec 2020 22:12:11 +0100

> Is it possible to import a Python library, such as graphviz, into an Ada package?

gnatcoll.python + a lot of binding work

From: Roger Mc <rogermcm2@gmail.com>
Date: Fri, 4 Dec 2020 13:19:10 -0800

> gnatcoll.python + a lot of binding work

I have been trying to figure out how to use gnatcoll.python. Unfortunately it doesn't seem to provide any supporting documentation.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 5 Dec 2020 00:17:12 +0100

> Unfortunately it doesn't seem to provide any supporting documentation.

What about this:
https://docs.adacore.com/gnatcoll-docs/scripting.html

I have a rudimentary Python bindings independent of GNATColl, which I use to run Python scripts from Ada. They were designed to load Python dynamically, I did not want to make the application dependent on Python installed. If you want, you can use them as a template. There is no documentation, but the code using them. But as I said, better not... (:-))

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 5 Dec 2020 10:38:10 +0100

> I would really appreciate seeing your "rudimentary Python bindings ".

Download sources of this:

http://www.dmitry-kazakov.de/ada/max_home_automation.htm

The project is large. Only these packages are related to Python:

1. Py is the bindings
2. Py.Load_Python_Library is an OS-dependent part for loading Python dynamically from a DLL (Linux or Windows)
3. Py.ELV_MAX_Cube is an implementation of a Python module in Ada. I.e. calling Ada from Python.
4. MAX_Control_Page contains a task that periodically runs a Python script. I.e. calling Python from Ada.

Ada Practice

Logging and Protected Actions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: Re: is there a version of unix written in Ada
Date: Thu, 1 Oct 2020 11:28:10 +0200
Newsgroups: comp.lang.ada

[...] BTW, I still do not know how to design an Ada-conform tracing/logging facility such that you could trace/log from anywhere, protected action included, and without knowing statically which protected object is involved.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 11:59:58 +0200

> BTW, I still do not know how to design an Ada-conform tracing/logging facility such that you could trace/log from anywhere [...]

Did you have a look at package Debug?

(https://www.adalog.fr/en/components#Debug)

It features, among others, a trace routine which is guaranteed to not be potentially blocking.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 12:21:46 +0200

> It features, among others, a trace routine which is guaranteed to not be potentially blocking.

It calls a protected operation on a different protected object, yes, this is non-blocking, and I considered the same, but is this legal? Maybe I am wrong, but I have an impression that walking away to another object is not OK. Or is that limited to protected entries only?

Another issue is having two different calls: Trace and protected Trace. If one is used instead of another, you have a ticking bomb in the production code. I remember that there was a GNAT pragma to catch it, but it was a run-time check, so it just replaced one type of explosive with another.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 1 Oct 2020 14:38:27 +0300

> It calls a protected operation on a different protected object, yes, this is non-blocking [...], but is this legal?

Yes.

If the program is using ceiling-priority-based protection, the priority of the calling object must be less or equal to the priority of the called object.

> Or is that limited to protected entries only?

An entry call is potentially blocking and therefore not allowed in a protected operation.

> Another issue is having two different calls: Trace and protected Trace. If one is used instead of another, you have a ticking bomb in the production code.

I assume that is a "feature" of the referenced Debug package, not of the basic method it uses to implement a logging facility.

I haven't looked at the Debug package, but I would have suggested a logging facility that consists of:

1. A FIFO queue of log entries implemented in a protected object of highest priority. The object has a procedure "Write_Log_Entry".
2. A task that empties the FIFO queue into a log file. The task calls an entry of the FIFO protected object to get a log entry from the queue, but executes the file-writing operations in task context, not in a protected operation.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 13:48:26 +0200

> I remember that there was a GNAT pragma to catch it, but it was a run-time check

Well, just use AdaControl with the rule:

```
check Potentially_Blocking_Operations;
```

;-)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 14:51:54 +0200

> If the program is using ceiling-priority-based protection, the priority of the calling object must be less or equal to the priority of the called object.

My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

If so, then let there be protected procedure P1 of the object O1 and P2 of O2. If P1 and P2 call to P3 of O3 that would be a problem. Ergo either wandering or concurrent protected protected calls must be illegal.

> 1. A FIFO queue of log entries implemented in a protected object of highest priority. The object has a procedure "Write_Log_Entry".

Yes, that was what I thought and what Debug.adb does. However Debug.adb allocates the body of the FIFO element in the pool. I would rather use my implementation of indefinite FIFO which does not use pools. I don't want allocators/deallocators inside protected stuff.

> 2. A task that empties the LIFO queue into a log file.

A simpler approach is to flush the queue by the first call to an unprotected variant of Trace. I believe Debug.adb does just this.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 1 Oct 2020 16:18:58 +0200

> My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

No. Protected objects guarantee that only one task at a time can be inside (ignoring functions). Multi-cores don't come into play.

> I don't want allocators/deallocators inside protected stuff.

As surprising as it may seem, allocators/deallocators are NOT potentially blocking operations. But I understand your concerns...

> A simpler approach is to flush the queue by the first call to an unprotected variant of Trace. I believe Debug.adb does just this.

Yes. Moreover, there is a Finalize of a controlled object to make sure that no trace is lost if the program terminates without calling any (unprotected) Trace.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 1 Oct 2020 18:38:12 +0300

> My mental picture was protected procedure calls executed concurrently on different cores of a multi-core processor. Would that sort of implementation be legal?

If the protected procedures belong to different protected objects, yes it is legal. But not if they belong to the same object, as J-P noted.

Note that the ordinary form of the ceiling-priority-locking method does not work for multi-cores, because a task executing at the ceiling priority of a protected object does not prevent the parallel execution of other tasks (on other cores) at the same or lower priority.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Oct 2020 19:37:01 +0200

[...] Now let's continue the example. What happens when the calling paths are:

O1.P1 --> O3.P3 --> O2.Q
O2.P2 --> O3.P3 --> O2.Q

Let Q1.P1 blocks Q2.P2 on an attempt to enter O3.P3:

O1.P1 --> O3.P3
O2.P2 --> blocked

Then O3.P3 calls O2.Q:

O1.P1 --> O3.P3 --> O2.Q
|
O2.P2 --> blocked V

This will either re-enter O2 or deadlock.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:10:10 -0500

[...]

A task has to wait to get access to a PO. This is *not* blocking, it is not allowed to do anything else during such a period. (This is why protected operations are supposed to be fast!). It's canonically implemented with a spin-lock, but in some cases one can use lock-free algorithms instead.

For a single core, one can use ceiling locking instead (and have no waiting), but that model seems almost irrelevant on modern machines.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:13:14 -0500

> [...] you have a problem when two independently running protected procedures of *different* objects call a procedure of a third object. You must serialize these calls, and that is effectively blocking.

Not really: blocking implies task scheduling (and possible preemption and priority inversion), whereas no scheduling happens on a protected call. There's just a possible wait. It's a subtle difference, admittedly, but it makes a world of difference to analysis.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 2 Oct 2020 07:36:07 +0200

To continue on Randy's response: mutual exclusion is not blocking. "Blocking" (as in "potentially blocking operation") means "being put on a queue", i.e. when the waiting time is potentially unbounded. The waiting time due to mutual exclusion is bounded by the execution time of the protected operation, and then can be included in the execution time of the waiting task. (In reality, it can be slightly more complicated, but the idea is that it is bounded).

[...]

In summary, the model of PO is two levels:

1) mutual exclusion, which is not "blocking"

2) for entries: queuing, which is "blocking"

Once you realize this, it should make this whole thread clearer....

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Oct 2020 08:56:38 +0200

> mutual exclusion is not blocking. "Blocking" (as in "potentially blocking operation") means "being put on a queue", i.e. when the waiting time is potentially unbounded.

It would be a poor definition, because deadlock is not bounded as well. If jumping from one protected object to another is legal, we can construct a deadlock out of mutual exclusion. We also have a situation when multiple tasks executing protected procedures are awaiting their turn to enter a procedure of some object. They will continue (if not deadlocked) in some order, which is obviously a queue.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 2 Oct 2020 09:42:02 +0200

> It would be a poor definition, because deadlock is not bounded as well. If jumping from one protected object to another is legal, we can construct a deadlock out of mutual exclusion.

But this would necessarily involve an "external call to the same protected object", which is defined as a potentially blocking operation. Note that AdaControl is quite powerful at detecting that situation (by following the call graph).

> We also have a situation when multiple tasks executing protected procedures are awaiting their turn to enter a procedure of some object. They will continue (if not deadlocked) in some order, which is obviously a queue.

No, it can be implemented with a spin lock. It is bounded by the number of waiting tasks x service time. You don't have to wait for some unpredictable barrier.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Oct 2020 22:14:49 -0500

> But this would necessarily involve an "external call to the same protected object", which is defined as a potentially blocking operation.

Note that such an operation doesn't really block, it is a deadlocking operation; Ada lumped it into "potentially blocking" in order to save some definitional overhead. (A mistake, in my view, it should simply have been defined to raise Program_Error or maybe Tasking_Error.) "Potentially blocking", in normal use, means something else.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Oct 2020 17:21:01 -0500

> O2.P2 --> O3.P3 --> O2.Q

This latter path is always going to deadlock, since the second call to O2 is necessarily an external call (you're inside of O3, not O2). An external call has to get the lock for the protected object, and since the lock is already in use, that will never proceed.

[If O3 was nested in O2, then the second call to O2 could be internal. But in that case, the first path would be impossible as O1 could not see O3 to call it.]

Remember that the decision as to whether a call is internal or external is purely syntactic: if a protected object is given explicitly in the call, one needs to trigger the mutual exclusion mechanisms again. The only time one doesn't need to do that is when the call does not include the object (that is, directly from the body of an operation).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Oct 2020 08:55:56 +0200

> This latter path is always going to deadlock, since the second call to O2 is necessarily an external call

Is that implementation or requirement? The lock can be task-re-entrant.

> Remember that the decision as to whether a call is internal or external is purely syntactic: if a protected object is given explicitly in the call, one needs to trigger the mutual exclusion mechanisms again.

Even when the object in the call is statically known to be the same?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Oct 2020 22:09:19 -0500

> Is that implementation or requirement? The lock can be task-re-entrant.

Language requirement. An external call requires a separate mutual exclusion. If Detect_Blocking is on, then Program_Error will be raised. Otherwise, any pestilence might happen.

> Even when the object in the call is statically known to be the same?

Yes. An external call *always* gets the lock again. I believe that was made the rule to make it obvious as to what will happen based on the form of call.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Oct 2020 08:42:03 +0200

> Yes. An external call *always* gets the lock again. I believe that was made the rule to make it obvious as to what will happen based on the form of call.

I mean this:

```
protected body O is
  procedure P1 is
  begin
    ...
  end P1;
  procedure P2 is
  begin
    P1;   -- OK
    O.P1; -- Deadlock or Program_Error
  end P2;
end O;
```

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 3 Oct 2020 10:44:59 +0300

> I mean this:
>
> protected body O is
> procedure P1 is
> begin
> ...
> end P1;
> procedure P2 is
> begin
> P1; -- OK
> O.P1; -- Deadlock or Program_Error

That is an internal call, so no deadlock nor error.

See RM 9.5(4.e), which is this exact case.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Oct 2020 10:16:15 +0200

> That is an internal call, so no deadlock nor error.

I.e. it is *not* based on the syntax of the call.

Anyway the rather disappointing result is that protected procedures may deadlock (or Program_Error) in a legal program.

So my initial disinclination to jump from one protected object to another is reasonable advice. Or at least the order in which protected objects are navigated must be the same.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 3 Oct 2020 13:44:47 +0300

> I.e. it is *not* based on the syntax of the call.

At least not on /that/ syntactical difference.

> Anyway the rather disappointing result is that protected procedures may deadlock (or Program_Error) in a legal program.

Legal programs can run into all sorts of problems, starting with use-before-elaboration.

> So my initial disinclination to jump from one protected object to another is reasonable advice.

Quite conservative advice, though.

> Or at least the order in which protected objects are navigated must be the same.

I would say that it is advisable to arrange the POs (or PO types) in a layered architecture and make inter-PO calls only from a higher-layer PO to a lower-layer PO.

GDNative Thick Binding Design

From: Michael Hardeman <mhardeman25@gmail.com>
Subject: GDNative thick binding design
Date: Thu, 15 Oct 2020 14:08:19 -0700
Newsgroups: comp.lang.ada

I'm working on a binding to the Godot game engine for Ada.

Project link here: https://github.com/MichaelAllenHardeman/gdnative_ada

Once the game engine has loaded your dynamic library it will call the function *_nativescript_init (where * is the symbol_prefix defined in the library resource config file). This function is responsible for registering objects, object methods, and allocating any memory needed.

What I want to discuss here is that I'm a bit at a loss as to how to design a thick binding wrapper around this object registration pattern. So let me describe the pattern.

I have a very simple example translated from C using the thin binding here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/master/examples/gdnative_c_example/src/simple.adb#L44

The objects must have a name, but may or may not override the constructor/destructor life cycle functions (which you pass in during registration)

There are

https://docs.godotengine.org/en/stable/classes/class_object.html#class-object

There is kind of a hierarchy at play as well:

the Node type extends Object

https://docs.godotengine.org/en/stable/classes/class_node.html#node

and has addition life cycle events like _process (callback on each frame) https://docs.godotengine.org/en/stable/classes/class_node.html#class-node-method-process

Now I don't even know where to start defining something nice in Ada that would match this pattern and would hide all the nastiness from the C binding. I kind of want the tagged record hierarchy structure, with overriding functions, but it should only register methods with godot you've overridden. How would I know what methods have been overridden? I feel like I need some kind of generic or helper functions?

I'm hoping some more experienced people might have some suggestions?

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 07:38:05 +0100

> I'm working on a binding to the Godot game engine for Ada.

Ok, so what you have now is a gcc generated binding, which isn't the nicest to work with.

What you really need to do is to start by wrapping up the thin inside a thick binding such that the plug-ins only use the thick binding and that any of the calls such as simple_constructor are wrapped, i.e.

Godot.Make(Instance : Godot.Root_Class; parameters...) -> calls simple_constructor(Instance.Internal_Pointer, parameters). Use overloads for this kind of stuff.

The way I bind to C is like this:

1) If it's a simple function that takes no parameters and returns nothing, then bind directly.

2) If it's a simple return type, use an expression function to bind.

3) Anything else gets a thick binding.

4) Types are mapped onto the C ones, so I lift out the definition from the thin binding and put it in the root package of the thick. I also rename so there's less repetitive stuff like GODOT_VARIANT_* and I case properly, this will be difficult for situations where identifiers are Ada keywords, so rename to something else completely if you have to, just document the change.

Essentially you want all the C nastiness inside the thick binding.

Look at SDLAda for some ideas, but this was done by hand. Anything generated by GCC needs to be hand massaged to be nicer imo.

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Fri, 16 Oct 2020 09:39:17 -0700

Thanks for the detailed reply. Unfortunately I think I didn't get my question across correctly.

I'm pretty familiar with most of the basic stuff I can do in Ada. I'm not asking for general advice on making a thick binding, I'm asking for help with one specific data structure/pattern.

What is the best way to make Ada types/functions that wrap a particular thing:

I just pushed a work in progress branch where you can see what I'm struggling with:

https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L29

https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/example_object.adb#L90

Is it possible to create a type (tagged record maybe) whose dispatching methods automatically register in some way?

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 19:09:13 +0100

> Is it possible to create a type (tagged record maybe) who's dispatching methods automatically register in some way?

If you mean call Register(Context); on construction of the object, then have you looked at the factory stuff?

http://www.ada-auth.org/standards/12rm/html/RM-3-9.html#I2118

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Fri, 16 Oct 2020 11:28:30 -0700

not when the object is constructed. I was wondering if something like the following were possible:

```
package GDNative.Thick.Objects is
  type Object is abstract tagged private;
  -- create abstract or null subprograms for
  -- each subprogram here:
  -- https://docs.godotengine.org/en/stable/
  -- classes/class_object.html#class-object
  function Name (Self : in Object'class)
return Wide_String is abstract;
  procedure Initialize  (Self : in out
              Object'class) is null;
  -- etc...
private
  type Object is abstract tagged null
record;
end;
```

But I need some way of knowing here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L28 what all the types that extend that object tagged type are, and what all the null methods they've chosen to override are. Kind of like the Java Class() style introspection.

I'm sure there must be some way of doing it better tho, with generics? I'm just not creative enough to see the solution atm.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 16 Oct 2020 19:31:56 +0100

> But I need some way of knowing here: https://github.com/MichaelAllenHardeman/gdnative_ada/blob/feature/adventure_game/examples/adventure_game/src/engine_hooks.adb#L28

That's what the generic constructor would allow.

> what all the types that extend that object tagged type are, and what all the null methods they've chosen to override are. Kind of like the Java Class() style introspection.

>

> I'm sure there must be some way of doing it better tho, with generics? I'm just not creative enough to see the solution atm.

You can't know what the null methods are. Why do you even need to know?

I'm probably not understanding this, tbf.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sat, 17 Oct 2020 03:09:04 -0700

> package GDNative.Thick.Objects is

> type Object is abstract tagged private;

>

> -- create abstract or null subprograms for each subprogram here:

> -- https://docs.godotengine.org/en/stable/classes/class_object.html#class-object

> function Name (Self : in Object'class) return Wide_String is abstract;

> procedure Initialize (Self : in out Object'class) is null;

> -- etc...

>

> private

> type Object is abstract tagged null record;

> end;

I do not know what you are trying to do, but I see a basic misunderstanding here wrt keyword abstract on operations. It has two fundamentally different purposes:

* When used on a primitive operation of a non-tagged type, it makes an inherited

operation disappear, i.e. this operation does no longer exist, e.g.:

```
type T is range -42..42;
function "/" (L, R: T'Base) return T'Base
        is abstract;
```

* When used on a primitive operation of a tagged type, this operation is dispatching and must be overridden for derived types; e.g.

```
type T is abstract tagged private;
procedure Op(X:T) is abstract;
type T1 is new T with private;
procedure Op(X:T1);
```

Now your

```
function Name (Self : in Object'class)
        return Wide_String is abstract;
```

is a classwide operation, not a primitive operation, so it cannot be overridden. It is not a primitive operation of any type, so it just declares that such an operation cannot exist - a rather useless declaration.

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Sun, 18 Oct 2020 22:21:38 +0200

My usual path is:

1) find bindings in other languages and try to understand their intention.

2) Generate a 1:1 binding to the C API since that will provide a sound ground (this is an 100% automatic process).

3) Write the high-level binding trying to mimic other language bindings while keeping an Ada twist to it,

With a minor effort I managed to do step one and two but step three is the hard one. Have a look on https://github.com/Ada-bindings-project/godot

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 21 Oct 2020 07:59:00 +0100

>> I'm probably not understanding this, tbf.

>

> Can you explain the Generic Constructor some more? I need to use it now, but I can't exactly figure it out. I found this example: https://www.adacore.com/gems/ada-gem-19 but I have no idea how they can use the 'Input attribute as the constructor function. It doesn't match the signature requested by the generic at all.

>

> I have a simple example I was trying to get working: https://ideone.com/f5bpr9

> Do you think you could help me understand where I'm going wrong here?

>

I've never used it, but this might help https://www.adaic.org/resources/add_content/standards/05rat/html/Rat-2-6.html

From: Michael Hardeman <mhardeman25@gmail.com>
Date: Sun, 25 Oct 2020 20:38:36 -0700

https://github.com/MichaelAllenHardeman/gdnative_ada

I've done an initial pass on the thick binding. [...]

Still, as is, it's pretty nice to use. It only takes just a tiny bit of user code to get an object registered and running a function on each frame.

https://github.com/MichaelAllenHardeman/gdnative_ada/tree/master/examples/adventure_game/src

Windows GUI Frameworks

From: DrPi <314@drpi.fr>
Subject: Which GUI framework?
Date: Thu, 29 Oct 2020 19:48:36 +0100
Newsgroups: comp.lang.ada

I'd like to create a PC (Windows) GUI program. This program needs to be able to create many Windows and tabs in one of them. A working thread receives data from a serial line and sends messages to the GUI to print received content.

I know the most common way is to use GtkAda. The problem is I'm an Ada beginner and I never used Gkt. So, the effort is double.

I have a quite good knowledge of wxWidgets since I have used wxPython for years. I thought I could use wxAda but it seems the project is dead.

Any other binding to wxWidgets that I'm not aware of?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 29 Oct 2020 20:23:55 +0100

> Any other binding to wxWidgets that I'm not aware of?

If that is only Windows (are you serious?), you do not need any. Simply use Windows GDI API directly. They are callable from Ada more or less out of the box because Windows handles all objects internally as graphic resources.

There are Win32Ada thin bindings, but it is incomplete and most of the time you do not need it.

The Microsoft's way of defining and using types is so idiotic that no reasonably usable thin Ada bindings are possible. I just declare an Ada counterpart new as appropriate with parameters of types I want in order to avoid casting types.

In short, Windows GDI is ugly but it is native and task-safe. (GtkAda is neither)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 29 Oct 2020 15:45:09 -0500

> If that is only Windows you do not need any. Simply use Windows GDI API directly. [...] There are Win32Ada thin bindings, but it is incomplete and most of the time you do not need it.

For Win32, both Claw (www.rrsoftware.com) and GWindows provide thick Ada bindings. Much easier to use than raw Win32.

From: DrPi <314@drpi.fr>
Date: Fri, 30 Oct 2020 10:37:13 +0100

> If that is only Windows (are you serious?),

Did I say that? ;)

I currently do my dev on a Windows machine but a cross-platform framework is welcome.

> In short, Windows GDI is ugly but it is native and task-safe. (GtkAda is neither)

Windows GDI... I used it a long time ago. Not my best memory.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 30 Oct 2020 09:52:11 +0000

> know the most common way is to use GtkAda. The problem is I'm an Ada beginner and I never used Gtk. So, the effort is double.

Gtk isn't all that pleasant either.

> I have a quite good knowledge of wxWidgets since I have used wxPython for years. I thought I could use wxAda but it seems the project is dead.

Yup, I agree that wxWidgets is much simpler as it was based on MFC, only portable.

At this time wxAda is dead on my hdd right now and not going to be resurrected until I get some money coming in.

> Any other binding to wxWidgets that I'm not aware of?

No, both efforts were abandoned as it was too much work. I have a start to a generator, but like I said, it's not happening right now.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 30 Oct 2020 10:54:54 +0100

> I currently do my dev on a Windows machine but a cross-platform framework is welcome.

Cross-platform would be:

1. GTK (GtkAda)

2. Qt (not sure about the project name)

3. HTTP (Gnoga)

From: Chris M Moore <zmower@ntlworld.com>
Date: Fri, 30 Oct 2020 11:36:27 +0000

> Cross-platform would be:

Or Tk via https://github.com/simonjwright/tcladashell (or https://github.com/thindil/tashy but I've not used that).

From: Jeffrey R. Carter
Date: Fri, 30 Oct 2020 13:31:45 +0100

Gnoga (https://sourceforge.net/projects/gnoga/) is all Ada (not a binding) and platform independent.

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 12:20:41 +0100

Gnoga is very interesting when the GUI is remotely run.

I think using such a system locally is nonsense (very resource hungry).

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 12:14:46 +0100

Binding to C++ libraries is a problem.

In the Python world, there are many ways to achieve this.

If I remember well, the author of wxPython has written its own binding system for version 3. Before version 3, he used a "standard" one but with many manual patches.

PySide (Python binding for Qt) authors also have written their own binding system after using one that was not fulfilling their needs.

It's a pity since I like wxWidgets' way of working.

From: DrPi <314@drpi.fr>
Date: Sat, 31 Oct 2020 17:30:15 +0100

Do you know SWIG (http://swig.org/)?

SWIG manages C++ bindings to many languages... but not Ada. However, SWIG tools might be of interest, like the tree parser outputting xml. Maybe SWIG can be modified to manage Ada. Just an idea. But not my skills.

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 31 Oct 2020 16:35:26 +0000

> Do you know SWIG (http://swig.org/)?

I know of it and no thanks. My generator would actually be simpler.

Publisher/Subscriber for Ada

From: DrPi <314@drpi.fr>
Subject: PubSub
Date: Sat, 31 Oct 2020 18:58:03 +0100
Newsgroups: comp.lang.ada

Another question indirectly concerning GUI programming:

Does an Ada "PubSub" package exist?

Something like this: https://pypubsub.readthedocs.io/en/v4.0.3/

Search on Alire returned no result.

Global search on the internet is "polluted" by many Ada answers.

From: Jeffrey R. Carter
Date: Sat, 31 Oct 2020 19:23:55 +0100

> Global search on the internet is "polluted" by many Ada answers.

There's Google custom search for Ada programming topics at https://thindil.github.io/adasearch/ and the Ada-specific search from the AdaIC at https://www.adaic.org/ada-resources/ada-on-the-web/

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 31 Oct 2020 19:38:09 +0100

> Another question indirectly concerning GUI programming: Does a Ada "PubSub" package exist?

Yes. We have a commercial middleware 100% in Ada. We use that thing in automation and control systems. Naturally, it provides publisher/subscriber services, distributed or not with controlled QoS. That is so to say horizontal communication between applications or tasks. It also has a vertical communication aspect abstracting hardware/protocols from application. E.g. you can publish/subscribe to a MQTT topic, or to an EtherCAT object, or to a CANOpen dictionary object etc without even knowing if that's really the thing, something else or another application.

Having said that, for horizontal communication inside a single process you do not need that in Ada. Many things done for other languages are not needed in Ada.

Ada protected objects and tasks provide much more efficient, safer (typed) and easier to use way to communicate between tasks.

From: DrPi <314@drpi.fr>
Date: Sun, 1 Nov 2020 11:36:37 +0100

> Ada protected objects and tasks provide much more efficient, safer (typed) and easier to use way to communicate between tasks.

What I'm looking for is not inter-task communication. It is some sort of message dispatcher (which is not thread safe). It is like a GUI event manager but for custom events.

A simple description here: https://wiki.wxpython.org/WxLibPubSub

This is very useful when using a GUI since it allows to directly send messages to windows/dialogs/controls.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 1 Nov 2020 12:18:20 +0100

> It is some sort of message dispatcher (which is not thread safe). It is like a GUI event manager but for custom events.

You do not need that stuff. Even less if that is not task safe. In the context of the same task, it is just a call. You need no marshalled arguments because the call is synchronous and it must be synchronous because it is the same task. The very term "event" makes no sense if the task that emits it is the task that consumes it.

> This is very useful when using a GUI since it allows to directly send messages to windows/dialogs/controls.

It is not useful, it is a mess, e.g. in GTK.

Anyway, the standard Ada library contains implementation of FIFO queues. If you want it 1-n rather than 1-1 use a blackboard instead of a FIFO.

Dueling Compilers

From: Jeffrey R. Carter
Subject: Dueling Compilers
Date: Wed, 25 Nov 2020 15:08:40 +0100
Newsgroups: comp.lang.ada

Consider the package

```
with Ada.Containers.Bounded_
Doubly_Linked_Lists;
generic
   type E is private;
package Preelaborable is
   package EL is new
         Ada.Containers.Bounded_
         Doubly_Linked_Lists (
         Element_Type => E);
end Preelaborable;
```

Two Ada-12 compilers give different results on this. Compiler G accepts it without problem. Compiler O rejects it with the error message preelaborable.ads: Error: line 6 col82 LRM:10.2.1(11.8/2), If a pragma Preelaborable_Initialization has been applied to the generic formal, the corresponding actual type must have preelaborable initialization AFAICT from the ARM, the generic formal Element_Type of Ada.Containers. Bounded_Doubly_Linked_Lists does not have pragma Preelaborable_Initialization applied to it. However, the type List, which probably has [sub]components of Element_Type, does.

Which compiler is correct? What is the intent of the ARM?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 25 Nov 2020 20:19:34 -0600

I'd say both compilers are wrong, in that the RM clearly has a bug here and one of the implementers should have complained about it to the ARG long ago. :-)

I'd suggest you post this question to Ada-Comment so that it gets on the ARG's radar.

(I'll call Preelaborable_Initialization "PI" in the following for my sanity. :-)

It's clear from 10.2.1 that a type with pragma PI which has components of a generic formal type has to have components that have a type with PI. It isn't possible to initialize such components without a function call, so the other possibility does not exist. The Bounded containers are designed such that there are components of the element type (more accurately, a component of an array of the element type). In order for there to be such a component, the formal type must have PI. Ergo, anybody for a bounded container written in Ada is necessarily illegal. This is a problem that someone should have brought up at the ARG.

Since it is not required to write language-defined package bodies in Ada, one could imagine that both compilers are correct in the sense that they are using some non-Ada language to implement the containers. But that is a fiction in the case of the containers (every implementation I know of is in Ada), and in any case, we intended the containers to be implementable in Ada. If they are not, that is a bug.

I don't know what the fix ought to be: adding PI to the formal private type would work, but it would reduce the usability of the containers in non-preelaborated contexts. Similarly, removing the PI from the container would work, but would reduce the usability of the containers in preelaborated contexts. Both seem pretty bad.

I'd be in favor of removing PI and Preelaboration in general from the language (it serves no purpose other than to encourage implementers to make optimizations that they should make anyway - the other intentions don't work or are better handled with other mechanisms), but I doubt that I'd get any support for that.

So this will have to be an ARG question -- I can't answer it definitively.

P.S. If you post this question to Ada-Comment, do me a favor and post this analysis along with it. That will save me having to reproduce it later.

From: Jeffrey R. Carter
Date: Fri, 27 Nov 2020 08:32:41 +0100

> Ergo, anybody for a bounded container written in Ada is necessarily illegal.

I think both compilers are doing macro-expansion of generics, so a generic is only really compiled when it is instantiated. Presumably any test code used actual parameters that the compiler could tell were PI, so they compiled OK.

> adding PI to the formal private type would work, but it would reduce the usability of the containers in non-preelaborated contexts. Similarly, removing the PI from the container would work, but would reduce the usability of the containers in preelaborated contexts. Both seem pretty bad.

I presumed that leaving PI on the container was an oversight.

> So this will have to be an ARG question -- I can't answer it definitively.

OK, I'll research the format of submissions to Ada-Comment and send it in.

> P.S. If you post this question to Ada-Comment, do me a favor and post this analysis along with it. That will save me having to reproduce it later.

I would have done that anyway. Thanks for confirming my suspicion that something is rotten in Denmark.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 27 Nov 2020 20:35:57 -0600

> I think both compilers are doing macro-expansion of generics, so a generic is only really compiled when it is instantiated.

That would be an incorrect implementation of generic units in Ada. One has to enforce the language rules only knowing the guaranteed properties of the formal types (knowing nothing about the actual). There is a later legality recheck in the specification of an instance, but that would be irrelevant in this case since the generic unit already is illegal.

> I presumed that leaving PI on the container was an oversight.

It definitely is intended, if the unit is Preelaborated, we definitely want any private types in it to be PI (lest they be unable to be used in Preelaborated units.

From: Jeffrey R. Carter
Date: Thu, 17 Dec 2020 21:22:50 +0100

For those who are interested, this became AI12-0409-1, approved 2020-12-09

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 18 Dec 2020 20:00:02 -0600

> For those who are interested, this became AI12-0409-1, approved 2020-12-09

For what it's worth, that approval included moving most of AI12-0399-1 to this AI, and making this AI a Binding Interpretation so it applies to Ada 2012 as well. We agreed not to require in the ACATS that implementations define the Preelaborable_Initialization aspect (if they have some other existing way to do this, that's fine by us for Ada 2012), but they can if they want. We will insist that bounded containers have P_I if the element type has P_I, and that they can be instantiated if the element type does not have P_I.

Advent of Code

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code
Date: Fri, 27 Nov 2020 19:12:21 -0800
Newsgroups: comp.lang.ada

Does anyone know about Advent of Code, and has anyone ever participated for Ada? It's typically a sequence of programming puzzles posed as an Advent calendar: one for each new day.

https://adventofcode.com/2020/about

Older examples are here:

https://adventofcode.com/2020/events

I had thought of it, but I don't have too much time. Some languages maintain their own mini-communities and leaderboards, and it might be a way to raise Ada's profile (or even SPARK'S?).

From: Jeremy Grosser
<jeremy@synack.me>
Date: Sat, 28 Nov 2020 19:36:48 -0800

I did Advent of Code in Ada last year. I got distracted by other projects and didn't finish it, but found it to be a very good way to learn with focused problems. My solutions are up on GitHub if you're curious, but knowing what I know now, they're far from optimal and some parts are definitely in need of refactoring.

https://github.com/JeremyGrosser/advent

From: Bojan Petrovic
<bojan_petrovic@fastmail.fm>
Date: Sun, 29 Nov 2020 15:03:45 +0100

I solved a couple of challenges from the last year's AoC in both Ada and Rust, just to get a feel for the differences between them in a puzzle solving context:

https://github.com/ALPHA-60/advent-of-code-2019

I've been organising a weekly recreational coding workshop at my company for the last couple of years, and we've been solving Project Euler and Codility tasks. I stopped doing it in March because of the Covid-19 situation, but we'll reboot it online on December 1st, when AoC 2020 starts, though our schedule will remain the same - one AoC problem per week.

A while ago we did some interview question exercises on #Ada Telegram group, so maybe we can do it again there.

From: John Perry <john.perry@usm.edu>
Date: Mon, 30 Nov 2020 23:08:29 -0800

Well, the first day wasn't too bad. It took me an hour, mainly because I'm not as familiar with Ada as I'd like. Once I re-learned file input & remembered the declare clause, it was quick.

I'll follow Jeremy Grosser's example and post my solutions to GitHub, too.

https://github.com/johnperry-math/AoC2020.git

From: Max Reznik <reznik@adacore.com>
Date: Tue, 1 Dec 2020 03:37:06 -0800

Someone posted on reddit: https://www.reddit.com/r/ada/comments/k4fn9w/anyone_else_participating_in_advent_of_code/

From: gautier_niouzes@hotmail.com
Date: Wed, 2 Dec 2020 12:51:04 -0800

Thanks John for the reminder about the Advent of Code. It's lots of fun!

Just before starting with today's puzzle, I had the idea of programming the solution with HAC (and the LEA editor). The quick edition-compilation-run cycle of HAC is an advantage for this contest. However, today, I was not quick enough to get points. Perhaps another day?

Links to my solutions are at the end of the following post: https://gautiersblog.blogspot.com/2020/12/advent-of-code-2020-with-hac-and-lea.html

From: Max Reznik <reznik@adacore.com>
Date: Wed, 2 Dec 2020 13:29:43 -0800

I gathered a list of GitHub repositories from this topic on a page, if someone wants to see all of them in one place.

https://github.com/reznikmm/ada-howto/tree/advent-2020

I also provided mine Ada solutions as Jupyter Notebooks. You can read them in Markdown or launch in the browser with "launch | binder" button.

Have fun :)

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 02 Dec 2020 14:59:18 -0800

> Just before starting with today's puzzle, I had the idea of programming the solution with HAC (and the LEA editor). The quick edition-compilation-run cycle of HAC is an advantage for this contest.

On these small files, can you really tell the difference in speed between GNAT and HAC? or (insert other favorite editor, mine is Emacs) and LEA? For me, everything is instantaneous.

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Mon, 14 Dec 2020 09:43:05 -0800

> On these small files, can you really tell the difference in speed between GNAT and HAC? or (insert other favorite editor, mine is Emacs) and LEA? For me, everything is instantaneous.

From GNAT Studio I get a range of 1.5 sec (an i5 PC @2.9 GHz) to 9 sec (a lightweight laptop) for building aoc_2020_12.adb (almost a benchmark for easy puzzles ;-)).

On the same source, I run hac -v2 aoc_2020_12.adb:

Compilation finished in 0.000335500 seconds.

Part 1: Manhattan distance of the ship to (0,0): 1631 (1631.0)

Part 2: Manhattan distance of the ship to (0,0): 58606 (58606.0)

VM interpreter done after 0.008894500 seconds.

So, for this kind of puzzle, it makes a difference (correct solution to part 1 was sent at 00:11:01).

But agreed, it's quite rare.

Especially on today's puzzle, I didn't even consider using HAC...

From: John Perry <john.perry@usm.edu>
Date: Mon, 14 Dec 2020 13:56:02 -0800

What follows is a long way of saying "Thank you." :-)

I spend about 2 hours on each puzzle, which probably doesn't speak well of my programming prowess (I've programmed for decades, so I can't really say it's because I'm learning Ada). Somehow I enjoy it enough to come back day after day.

The puzzles themselves are usually easy (to me), and most of the ones with a non-trivial solution can probably be solved trivially, with one exception. At least the mathematics has gotten a little more sophisticated; I used the Chinese Remainder Theorem recently, which I got a kick out of implementing in Ada as a one-line function (not including a support function to compute a modular inverse). I noticed that Maxim used Fermat's Little Theorem.

I sometimes roll my eyes at the puzzles, but the one thing I've really enjoyed so far is how each new puzzle has nudged me to learn a different Ada feature with each new puzzle. I'd spend a lot less time on it if I allowed myself to use a computer algebra system, but the point is to learn Ada, and the really nice surprise has been how people have helped out, some of them even commenting directly on GitHub.

Advent of Code Thread Compilation

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Advent of Code Thread Compilation
Date: Fri, 05 Feb 2021 17:59:27 +0100
To: Ada User Journal Readership

[This is a special message in that I am directly writing it to the Ada User Journal readership. Besides the previous thread on Advent of Code, there were a number of threads for each day. These threads refer to unstated off-groups problems and the discussion is too informal and disjointed to make a coherent post-hoc read, even after summarizing. For that reason, I am not including these threads as-is in the Digest. For the interested readers, I have compiled all the related threads in the newsgroup at the end of this message.

There are nonetheless some interesting tidbits and snippets discussing Ada features, libraries and resources that, even without context, may be useful pointers to follow. I am keeping these in the following messages, with the title of the thread they belong to. —arm]

Day 2: https://groups.google.com/g/comp.lang.ada/c/ASTsQiya1yQ/m/sx27Sb3XAgAJ

Day 3: https://groups.google.com/g/comp.lang.ada/c/zsZV1RSf01c/m/Fl7CTEB2AAAJ

Day 4: https://groups.google.com/g/comp.lang.ada/c/7CmcyU37SkA/m/aI2k3YxfAwAJ

Day 5: https://groups.google.com/g/comp.lang.ada/c/aOF1sirDOiY/m/GEDagaqpAwAJ

Day 6: https://groups.google.com/g/comp.lang.ada/c/co9hjh6F1Ng/m/xbdMecnjAwAJ

Day 8: https://groups.google.com/g/comp.lang.ada/c/jxx-4c2hPng/m/3EO7rO30BAAJ

Day 10: https://groups.google.com/g/comp.lang.ada/c/Z4mmw_t94Ls/m/X2MG3IDfAQAJ

Day 11: https://groups.google.com/g/comp.lang.ada/c/BIBRIl7iirw/m/1tO_250LAgAJ

Day 12: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 17: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 19: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 23: https://groups.google.com/g/comp.lang.ada/c/lqb0iuLXm5E/m/FVGVnyNlAgAJ

Day 25: https://groups.google.com/g/comp.lang.ada/c/zcMzC_q9KmA/m/Aa7iA3q4BAAJ

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code Day 2
Date: Wed, 2 Dec 2020 15:45:25 -0800
Newsgroups: comp.lang.ada

> ...I should have used Gnatcoll.regexp.

I was wondering if there was a pattern matching library I could use, and had wanted to ask that, but forgot.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 03 Dec 2020 03:52:47 -0800

'Reduce is a new Ada 2020 attribute

(www.ada-auth.org/standards/2xrm/html/RM-4-5-10.html); it can sum an array.

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code Day 3
Date: Sat, 5 Dec 2020 07:11:06 -0800

> Day 4 task is dull :)

>

> https://github.com/reznikmm/ada-howto/blob/advent-2020/md/04/04.md

Flourishes like this:

```
return Passport (byr .. pid) =
          (byr .. pid => True);
```

illustrate idioms that I really want to learn, thanks for sharing.

From: John Perry <john.perry@usm.edu>
Subject: Advent of Code day 5
Date: Sat, 5 Dec 2020 09:57:00 -0800

According to the Internet (And Therefore It Is True (TM)) the A380 can seat up 853 people. My problem had up to 894 seats, with the first 5 missing, so it wasn't that far beyond the realm of reason.

Then again, I don't know if anyone would want to fly an A380 configured for 853 people.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code day 5
Date: Sun, 06 Dec 2020 08:21:24 -0800

> and ran it through cut/sort/uniq

Next time, try ada.containers.generic_array_sort;

http://www.ada-auth.org/standards/2xrm/html/RM-A-18-26.html

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code day 5
Date: Sun, 06 Dec 2020 08:27:54 -0800

> Next time, try ada.containers.generic_array_sort;

> http://www.ada-auth.org/standards/2xrm/html/RM-A-18-26.html

Or Doubly_Linked_Lists.Generic_Sorting:

http://www.ada-auth.org/standards/2xrm/html/RM-A-18-3.html

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Advent of Code Day 7
Date: Mon, 7 Dec 2020 17:44:44 -0600

> Entry: Bag_Entry := (Quantity => 10);

>

> However, GNAT says this is invalid [...]

In Ada 2005 and later, write:

```
Entry: Bag_Entry := (Quantity => 10,
          Description => <>);
```

In an aggregate, <> means a default initialized component. Following the Ada Way TM ;-), one has to explicitly ask for a default initialized component - just leaving it out might have been a mistake or intended -- neither the compiler nor a reader can tell. The above is clearly intended.

From: Jeffrey R. Carter
Subject: Advent of Code Day 7
Date: Tue, 8 Dec 2020 12:25:54 +0100

>

> type Bag_Entry is record

> Description: Bag_Description := "";

Humans are notoriously bad at counting things, and even worse at counting things they can't see, so this kind of literal can be a source of errors, especially during modification. (At least with Ada these tend to be compiler errors, not run-time errors.)

Of course, Ada offers a Better Way. You can write

```
Description: Bag_Description :=
          (Bag_Description'range => ' ');
```

or

```
Description: Bag_Description :=
          (others => ' ');
```

and be proof against any changes to Bag_Description's bounds.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Advent of Code Day 10
Date: Fri, 11 Dec 2020 09:04:27 -0800

> My answer was able to fit in a Long_Long_Integer on my machine. But, due to a bug, I did play with the Big_Integers package. It worked well, and I'd recommend taking a look at it for upcoming

Yes; GNAT Community 2020 with -gnat2020 and -gnatX supports Ada.Numerics.Big_Integer. I updated my solution to use that.

From: Jeffrey R. Carter
Subject: Advent of Code Day 10
Date: Sat, 12 Dec 2020 23:25:41 +0100

> hmm. I got constraint error when I used Long_Integer; maybe that's not 64 bits? Using Ada.Big_Numbers.Big_Integers was a good exercise anyway.

That sounds like C thinking. If you need 64 bits, say so, don't hope that optional language-defined types will be big enough.

```
type S is range -(2 ** 63) + 1 .. 2 ** 63 - 1;
type U is mod 2 ** 64;
```

I used

```
type U is mod
          System.Max_Binary_Modulus;
```

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Advent of Code Day 17
Date: Fri, 18 Dec 2020 11:47:59 -0800

> Advent of Code hasn't been a *complete* waste of time. ;-)

Far from that: now the major part of the test suite for HAC stems from AoC:

[Omitted output of 27 successful tests for HAC, 15 of them being Advent of Code entries. —arm]

Starting Time of Real-time Clock

From: Simon Wright <simon@pushface.org>
Subject: Ada.Real_Time.Time_First
Date: Wed, 09 Dec 2020 12:30:44 +0000
Newsgroups: comp.lang.ada

I opened an issue[1] on Cortex GNAT RTS, saying

> You'd expect Ada.Real_Time.Time_First to be quite a long time before any possible value of Ada.Real_Time.Clock; but in fact the system starts with Clock equal to Time_First.

On the other hand, I had written

```
Last_Flight_Command_Time :
Ada.Real_Time.Time
:= Ada.Real_Time.Time_First;
...
Quad_Is_Flying :=
  Ada.Real_Time.To_Duration (Now -
    Last_Flight_Command_Time)
  < In_Flight_Time_Threshold;
```

but Now - Last_Flight_Command_Time is going to be quite small, to start with, so Quad_Is_Flying is going to be True when it shouldn't be.

The workaround I used was

```
Quad_Is_Flying :=
  Last_Flight_Command_Time /=
  Ada.Real_Time.Time_First
```

and then

```
Ada.Real_Time.To_Duration (Now -
  Last_Flight_Command_Time)
  < In_Flight_Time_Threshold;
```

In other words, I was using Time_First as a flag to indicate that Last_Flight_Command_Time was invalid.

What would your standard pattern for this sort of problem be? Especially considering that if I make Time_First a large negative number I'll get the opposite problem, e.g. predicting ahead for a very large interval, possibly even leading to numeric overflows.

I'm thinking of a Time type with the concept of validity, possibly built round

```
type Time (Valid : Boolean := False) is
record
  case Valid is
    when True  => Value :
          Ada.Real_Time.Time;
    when False => null;
  end case;
end record;
```

and addition, etc. with appropriate preconditions.

(not so sure about the discriminated record, might be more trouble than it's worth)

[1] https://github.com/simonjwright/*cortex-gnat-rts/issues/33

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 9 Dec 2020 14:16:10 +0100

> What would your standard pattern for this sort of problem be?

I would use Next_Time instead of Last_Time:

```
Next_Flight_Command_Time : Time :=
Time_First;
begin
  loop
    Now := Clock;
    if Now >= Next_Flight_Command_Time
then
      Fire_All_Rockets;
      Next_Flight_Command_Time :=
        Next_Flight_Command_Time +
        In_Flight_Time_Threshold;
    end if;
  end loop;
exception
  when Constraint_Error =>
   -- the End of Times!
    Put_Line ("Thank you for your
              cooperation!");
    Fire_Death_Star;
    Self_Destroy;
end;
```

From: Simon Wright <simon@pushface.org>
Date: Wed, 09 Dec 2020 20:07:32 +0000

> I would use Next_Time instead of Last_Time:

Great idea; the name isn't right in my context, but the method applies very well. (It's the time by which the next flight command has to have been given before we decide we're not flying anymore. I plead that (a) this logic seems not to be our Earth logic, (b) it's a translation from someone's C, (c) the original code has a comment expressing doubt)

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 9 Dec 2020 16:21:02 +0200

> I opened an issue[1] on Cortex GNAT RTS, saying

>

> You'd expect Ada.Real_Time.Time_First to be quite a long time before

> any possible value of Ada.Real_Time.Clock; but in fact the system

> starts with Clock equal to Time_First.

I don't see any reason for expecting Time_First to be far in the past relative to program start. In fact, RM D.8(19) says "For example, [the start of Time] can correspond to the time of system initialization".

Contrariwise, it could be useful to know that Clock actually starts from Time_First, because I have often needed a "Start_Time" object that records the Clock at the start of the program, and it would be much simpler to use Time_First, if Time_First is known to equal the initial Clock.

> Quad_Is_Flying :=

> Ada.Real_Time.To_Duration (Now - Last_Flight_Command_Time)

> < In_Flight_Time_Threshold;

If Time_First, as the initial value of Last_Flight_Command_Time, would really be in the far past compared to Now, that computation risks overflowing the range of Duration, which may be as small as one day (86_400 seconds), RM 9.6(27).

> The workaround I used was [...] I was using Time_First as a flag to indicate that Last_Flight_Command_Time was invalid.

Even that can still overflow Duration, if more than one day can pass since the last flight command.

> What would your standard pattern for this sort of problem be?

You have two problems: your assumption about Time_First (or perhaps it's not an assumption, if you make your own RTS) and the possible overflow of Duration.

To indicate an invalid Last_Flight_Command_Time, I would either use a discriminated type wrapping a Time value that depends on a Valid discriminant, as you suggested, or just have a Boolean flag, say Flight_Commands_Given that is initially False. I would use the discriminated type only if there is more than one such variable or object in the program.

For the overflow, I suggest changing the comparison to

```
Now < Last_Flight_Command_Time
    + To_Time_Span
     (In_Flight_Time_Threshold)
```

assuming that Last_Flight_Command_Time is valid in the sense we are discussing. That will overflow only when Last_Flight_Command_Time approaches Time_Last, and the program is likely to fail then anyway.

From: Simon Wright <simon@pushface.org>
Date: Wed, 09 Dec 2020 20:16:24 +0000

[...] This conversation has been very valuable, particularly in the case of other similar tests. I suspect, though, that "are we still flying?" is a question that'll take more thinking to resolve!

Possible to Recover Default Value of Scalar Type?

From: reinert <reinkor@gmail.com>
Subject: Possible to recover default value of scalar type?
Date: Sun, 13 Dec 2020 01:54:40 -0800
Newsgroups: comp.lang.ada

Assume the following code:

```
type A_Type is new Natural range 0..9 with
Default_Value => 9;
A : A_Type;
```

Is it later on here possible to get access to the default value (9)? If A was a component of a record, one could get it "9" via

```
some_record'(others =><>).A
```

But more directly? [Without declaring a variable, as is made clear in some omitted posts. —arm]

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Mon, 14 Dec 2020 01:01:21 -0800

I do not really understand the problem. It seems you want to be able to access the default value like so:

```
N: Natural := Natural(A_Type'Default_Value);
```

This is not possible. There is no corresponding attribute 'Default_Value.

If this presents a real problem, submit it to Ada comment stating why this is important.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 10:38:40 +0100

> If this presents a real problem, submit it to Ada comment stating why this is important.

It could in the cases like this:

```
procedure Library_Foo (Bar : Baz :=
        Baz'Default_Value)
```

You can declare constants in some places, but not at the library level. But in any case, being forced to declare a constant each time you need to get at the default value?

The same problem arises with container generics. If you have an array keeping container elements, logically freed elements need to be "destroyed" in some way. The default type value would be that thing as well as a default for Null_Element, if used.

I think that all non-limited types one could declare uninitialized, must have S'Default_Value equal to the default value the compiler would use. And it should produce same warnings uninitialized values do:

```
Put_Line (String (1..10)'Default_Value);
-- print garbage
```

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Mon, 14 Dec 2020 07:56:29 -0800

> procedure Library_Foo (Bar : Baz := Baz'Default_Value)

Suppose type Baz has no default value aspect. Then a call to Library_Foo without parameter would use what?

A solution could be that the attribute is illegal if there is no aspect. The compiler knows.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 17:31:29 +0100

> Suppose type Baz has no default value aspect. Then a call to Library_Foo without parameter would use what?

The default used by the compiler in this:

```
declare
   Bar : Baz;
begin
```

with an appropriate warning of course.

[It was a language design bug to allow implicitly uninitialized variables in the first place. Declarations like above should have been illegal.]

> A solution could be that the attribute is illegal if there is no aspect. The compiler knows.

I would argue that if

```
declare
   Bar : Baz;
begin
```

is legal, then it must be logically equivalent to:

```
declare
   Bar : Baz := Baz'Default_Value;
begin
```

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Dec 2020 18:24:54 +0000

> [It was a language design bug to allow implicitly uninitialized variables in the first place. Declarations like above should have been illegal.]

There is an argument that you should only initialise variables at the point of declaration if you know what value they should take; so that the compiler can detect the use of uninitialised variables.

If you always initialize variables, even if you don't know what value they should take, the compiler can't help you if you forget to assign the correct value.

Personally I always try hard not to declare an uninitialised variable.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Dec 2020 19:53:23 +0100

> There is an argument that you should only initialise variables at the point of declaration if you know what value they should take; so that the compiler can detect the use of uninitialised variables.

I think Robert Dewar argued that variables must be declared in the narrowest possible scope. Which would imply that at the beginning of that scope you should know the value, because it would be the first use of the variable.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:21:53 -0600

> N: Natural := Natural(A_Type'Default_Value);

We considered an attribute like that, but it becomes a semantic problem if the type doesn't have a Default_Value and you are in a context where you don't know (such as for a generic formal type). I vaguely remember some other semantic problem, but I don't remember the details. These things could be worked out, but it seemed messy.

I've long wanted <> to work as it does in aggregates generally (if that existed, I'd also have a restriction to require all objects to be initialized; that would provide an encouragement to initialize as many objects as possible; right now, the iffy thing (not initializing) is the easiest).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:26:10 -0600

> procedure Library_Foo (Bar : Baz := Baz'Default_Value)

I would have suggested to write this as:

```
procedure Library_Foo (Bar : Baz := <>)
```

since this is the syntax used in aggregates (and why should aggregates have all the fun??).

> Put_Line (String (1..10)'Default_Value); -- print garbage

The above isn't a legal attribute prefix in any case (can't slice a type). And you don't need to because this is clearly an aggregate (which is legal in Ada 2012):

```
Put_Line (String'(1..10 => <>));
-- print garbage
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Dec 2020 19:27:39 -0600

> The compiler knows.

Not always. Never forget generics. One would hope to be able to use this on generic formal types, as most of them are going to have default values (at least in new code).

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 15 Dec 2020 07:47:32 +0100

> I think Robert Dewar argued that variables must be declared in the narrowest possible scope.

Not applicable if your variable is used in a loop:

```
   V : Integer;
begin
   loop
      Get (V);
      exit when V =0;
      -- do something with V
   end loop;
```

Clearly, initializing V makes no sense.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Dec 2020 08:23:33 +0100

> Not applicable if your variable is used in a loop

```
   loop
      declare
         V : constant Integer := Get;
      begin
         exit when V = 0;
         -- do something with V
      end;
   end loop;
```

It is related to another long standing issue with returning values (multiple values) from functions and functions with in out parameters (resolved recently).

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Dec 2020 08:35:32 +0100

>> Put_Line (String (1..10)'Default_Value); -- print garbage

> The above isn't a legal attribute prefix in any case (can't slice a type).

I mean a subtype.

> And you don't need to because this is clearly an aggregate (which is legal in Ada 2012):

> Put_Line (String'(1..10 => <>)); -- print garbage

Yes, I would prefer the box notation too. However having a proper name would has some advantages too:

```
subtype S is T range T'Default_Value - 100..T'Default_Value + 100;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:43:56 -0600

> subtype S is T range T'Default_Value - 100..T'Default_Value + 100;

If box was generally allowed, you could qualify it to get this effect:

```
subtype S is T range T'(<>) - 100 .. T'(<>) + 100; -- Not Ada, but should be IMHO. :-)
```

and it's shorter, too. Of course, if T doesn't have a default value, neither of the above is a good idea. :-)

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 15 Dec 2020 10:07:02 +0100

> V : constant Integer := Get;

Well, you can push anything in a function, but it's not always clear/readable/simpler...

> V : Integer := <>; -- Invented syntax for explicit lack of initialization

That would make more sense: make initialization required, and say so if you don't care.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:48:06 -0600

> Clearly, initializing V makes no sense.

Saying that you *meant* to have an uninitialized value does make sense, though:

```
V : Integer := <>;
-- Not Ada, but should be IMHO.
```

Whenever something is omitted, one never knows whether it was on purpose or a mistake. You get similar issues when "else" is omitted (RR's style guide only allows that in very specific circumstances). It's unfortunate that Ada doesn't have a positive way to indicate default initialization, outside of aggregates.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Tue, 15 Dec 2020 10:14:59 -0800

Just a story about my work (long ago):

Our coding standard required for every type declaration a default value that indicated an uninitialised value:

```
type T is ...
Nd_T : constant T := ...;  -- Nd: not defined
X: T := Nd_T;  -- required
```

The idea was that this Nd value should be thus that it would be likely to produce an exception when used in an expression. Also any change of this value should have absolutely no effect on the code. In any case, at some time it was decided that the Nd value for numeric types was 0. The effect: It was no longer possible to see whether in a declaration like

```
X: T := Nd_T;
```

denoted a truly undefined value or a concrete and correct initial value.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2020 18:53:06 -0600

> It was no longer possible to see whether in a declaration [...] this value denoted a truly undefined value or a concrete and correct initial value.

Typically, values like this, at least those used in debuggers, use some permutation of 16#DEADBEEF# since it is obvious in data dumps, and is a rather unlikely value to be intended. The next version of Janus/Ada will initialize all "uninitialized" objects to this value unless you tell it not to. (Essentially, a version of Normalize_Scalars, except that these days it doesn't make much sense for that not to be the default. Optimization can remove most unneeded initializations, and if they are actually needed, it's better to have a known dubious value than stack garbage.)

Ada Syntax Questions

From: DrPi <314@drpi.fr>
Subject: Ada syntax questions
Date: Thu, 17 Dec 2020 23:39:44 +0100
Newsgroups: comp.lang.ada

Ada claims to have a better syntax than other languages. I'm fine with, but...

1) What about array indexing ?

In some other languages, arrays are indexed using square brackets. In Ada, parentheses are used for function calls and for array indexing. In the code "status := NewStatus(some_var);", you can't tell if NewStatus is a function or an array.

2) In Ada, a function without arguments is called without any parentheses.

In the code "status := NewStatus;", you can't tell if NewStatus is a function or a variable.

For my knowledge, are there good reasons for these syntaxes?

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Date: Thu, 17 Dec 2020 15:18:34 -0800

1) This allows you to replace your array with a function with the same name, which takes the subscript as an argument and returns a value, without touching your client code. Think about an expensive lookup table -vs- a simple function which computes your data. Do not see this as an ambiguity but rather a nice uniformity of calling something for a value.

2) Nearly the same, but in another context and without an argument. "NewStatus" could be, e.g., a constant, as long as types match.

From: Jeffrey R. Carter
Date: Fri, 18 Dec 2020 09:26:39 +0100

> 1) What about array indexing?

The requirements for the language included a restricted set of characters for source code that did not include brackets. So that is the primary reason parentheses are used.

However, both arrays and functions are often used as maps, and so an after-the-fact rationalization is that using the same syntax for both array indexing and function calls makes it easy to switch between the two.

> 2) In Ada, a function without arguments is called without any parentheses.

> In the code "status := NewStatus;", you can't tell if NewStatus is a function or a variable.

That's because Newstatus is a terrible name. If you'd used New_Status there would be no confusion.

Seriously: Ada 80 required empty parentheses for a subprogram call with no explicit parameters. During the review process that resulted in Ada 83, these were universally reviled and so were eliminated.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 18 Dec 2020 10:18:45 +0100

1. Separation of interface and implementation. Being an array or function is an implementation detail of a map or a named entity.

Another example is pointer dereferencing. In Ada X.A is the same as P.A. In C you have X.A vs P->A.

Yet another one. All instances of parameterization in Ada deploy () parentheses. In C++ it would be <>, [], (), depending on semantically irrelevant context.

2. Languages that like C use bottom-up matching are forced to distinguish certain

things prematurely on the syntax level. This is also the reason why you cannot use the result type to distinguish signatures in C++, but you can in Ada. Thus in C++ you would have something as disgusting as

```
123ull
```

while in Ada it is just

```
123
```

Long time ago anything but strictly bottom-up matching was considered too complicated or impossible. So artificial distinctions like () vs [] were invented and then promoted into orthodoxy.

From: Mart van de Wege <mvdwege@gmail.com>
Date: Fri, 18 Dec 2020 17:55:56 +0100

> 1) What about array indexing ?

Why would you care? It is obvious that NewStatus will return something based on the value of some_var. How it does that, by array dereference or function call should make no difference to the caller; they are only interested in the final value of status.

Or another look at it: array indexing is effectively a function call anyway. It is "return value of array_base + index".

> 2) In Ada, a function without arguments is called without any parentheses.

Again, why would you care how NewStatus returns a value? Either by returning the value of a function or by dereferencing a variable, all you're interested in is the value assigned to status.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 18 Dec 2020 18:38:27 +0100

> 2) In Ada, a function without arguments is called without any parentheses.

As others have stated, why do you care?

I often mock up a function with a constant, add a pragma compile_time_warning/error ("fix implementation later") and only later write the body of the function. And that is the only code change - I don't need to add an useless empty pair of () just because it is a function to all the callers

> For my knowledge, are there good reasons for these syntaxes?

Yes

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 18 Dec 2020 21:35:37 +0200

> Ada claims to have a better syntax than other languages.

I would say the claim is that the Ada syntax was rationally designed to have certain properties, which are desired by certain users (us Ada programmers) so it is "better" for us, although some aspects are subjective for sure.

In addition to what others have said, here are some further comments on

the examples you gave:

> 1) What about array indexing?

There are proposals to allow [] as well as (), mainly to increase familiarity for new Ada users.

> 2) In Ada, a function without arguments is called without any parentheses.

Parameterless functions are rare, and properly so.

Parameterless procedures are much more common. Writing

```
Frobnicate_Widget();
```

is longer than

```
Frobnicate_Widget;
```

and seems to have no advantages over the shorter form.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 18 Dec 2020 15:09:19 -0800

> 1) What about array indexing?

This is true.

You seem to be implying this is bad; why?

> 2) In Ada, a function without arguments is called without any parentheses.

This is true.

You seem to be implying this is bad; why?

> For my knowledge, are there good reasons for these syntaxes?

Yes. See the Ada Rationale: http://ada-auth.org/standards/rationale12.html

From: DrPi <314@drpi.fr>
Date: Sat, 19 Dec 2020 12:50:40 +0100

Thanks all for your answers.

> Why would you care?

Calling a function can have side effects. Accessing an array or a variable can't have side effects.

> You seem to be implying this is bad; why?

Reading the code can't tell you the writer's intentions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 13:40:25 +0100

> Calling a function can have side effects. Accessing an array or a variable can't have side effects.

Untrue. Both array and variable access have side effects on the registers, on the cache, on the process memory paging, in the form of exception propagation etc. Even direct effects on the outside world are possible when using machine memory load instructions. E.g. on some hardware reading memory at the specific address location means physical serial input.

All these effects are either desired parts of the implementation or else bugs to be fixed. If desired, why do you care?

> Reading the code can't tell you the writer's intentions.

What intentions? Unless you are talking about the intention to deploy a specific machine instruction, function or array gives you no clue. But even then. PDP-11 FORTRAN IV used subprogram calls to implement basically everything, elementary arithmetic operations. If the function is inlined, where is any call? Functions can be tabulated into lookup tables. Arrays can be compressed into functions.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sat, 19 Dec 2020 09:01:53 -0800

> Calling a function can have side effects. Accessing an array or a variable can't have side effects.

The declaration of the function is a contract about pre and post conditions, albeit in Ada incomplete. In SPARK, the contract is firm. As a user of the function, you have to believe the programmer that he follows the contract. If the implementation needs a side effect, so be it.

If on the other hand you are a maintainer or are chasing a bug, you have to check the requirements first, not the body of the function. This comes later.

> Reading the code can't tell you the writer's intentions.

The intentions are in the requirements (or in the accompanying comments, you hope they are up to date and not wrong). If there are none, good luck.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 09:13:56 -0800

> Untrue. Both array and variable access have side effects on the registers, on the cache, on the process memory paging, in the form of exception propagation etc.

Dmitry, DrPi here is referring to side-effects as viewed from the functional-programming paradigm's perspective. Some programming languages have a "pure" designator (usually the keyword: pure) that assures that this subroutine and all invoked subroutines therein are pure (i.e., have no FP side effects).

The side effects of which you speak are at the machine-code level: e.g., setting/clearing comparison flag(s), setting/clearing carry flag, setting/clearing overflow/underflow flag(s), evictions from L1/L2/L3 cache, (on RISC processors) latching an address in preparation of a load/store, and so forth. None of these are externally observable side effects from FP's perspective above the machine-code level. DrPi's FP goals are valid.

> > Reading the code can't tell you the writer's intentions.

> What intentions?

The intentions of the Ada programmer to design an overtly FP-pure or either an overtly FP-impure subroutine or an FP-impure subroutine by happenstance. Subroutine here is preferably a function, preferably at that a single-parameter function (for ability to utilize over a century of mathematical-analysis techniques). Ada is showing its 1970s vintage by unfortunately omitting overtly expressing FP pureness as a fundamental principle (among a few other FP features). DrPi's FP goals are valid.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 18:49:08 +0100

> The side effects of which you speak are at the machine-code level

Memory paging is pretty much observable.

What you are saying is a question of contracts. The contract must include all effects the user may rely on. The contract of a function may include observable effects or have none (to some extent).

If contracts were indeed relevant to the syntax then functions without contracted side effects must have been called using [] instead of ().

No? Then it is not about the contracts.

>>> Reading the code can't tell you the writer's intentions.

>> What intentions?

> The intentions of the Ada programmer to design an overtly FP-pure or either an overtly FP-impure subroutine or an FP-impure subroutine by happenstance.

Intentions are constraints expressed by contracts. Everything else is implementation details.

Ada programmers are not motivated by pureness of a subroutine. These are totally irrelevant. What is relevant is the strength of the contract. Functions without side effects are preferable just because they have weakest preconditions and strongest postconditions. Side effects weaken postconditions.

For the clients these are of no interest, even less to deserve a different syntax. The user must simply obey the contract whatever it be, ignoring the implementation as much as possible.

Ada's unified syntax is a great help here. I quite often replace arrays and variables with functions. It would be great if literals were fully equivalent to parameterless functions.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 10:40:53 -0800

> No? Then it is not about the contracts.

As witnessed by your final sentence quoted below and multiple other replies along this thread, the key tactical advantage of Ada's usage parentheses for array indexing is to accomplish a switcheroo days, weeks, months, years, or decades later: to substitute a function invocation later for what was formerly an array index. Cute trick. Advantageous in some situations. But for people like DrPi who seek contractual assurance of FP-purity of (all?) invoked functions (and overt declaration of impurity of other functions), Ada's 1) implicit switcheroo there in unfortunate combination with Ada's 2) lack of flamboyantly advertising impurity in the replacement function does in fact violate the purity portion of the contract that the mere offset-into-array implementation had—and indeed •overtly• declared in its specification as a mere offset-into-array operation-of-unquestionable-purity.

It is okay for a 1970s Ada to not foresee this, because FP was not a mainstream programming practice back then. (But, btw, it is not as okay for there to be a lack of HOLWGn each decade since the 1980s to revisit whether HOLWG1 forgot anything, where $n>1$, $n \in \mathbb{Z}$.) This 1970s faux pas in letting a silent slip-streamed switcheroo into the core contract-definition declaration mechanism of Ada (not comments! btw, tisk tisk) is merely some tarnish that an AdaNG (next-generation Ada) would fix: e.g., by mandating that all functions (and procedures?) shall be overtly declared & enforced to be pure or impure, which would then mean that only pure functions could substitute for array indexing is the ()-based switcheroo on which so many replies in this thread hang their hat. And DrPi would enjoy seeing the compile-time errors emerge when some cavalier programmer over yonder changed an array index to an •impure• function invocation as contract violation. The cute implicit switcheroo isn't evil, but the lack of compile-time detection of impurity in the switcherooed function is what is evil. (While drinking tea as none of my business as the meme goes,) I actually claim that Ada's usage of parentheses for array indexing was merely happenstance copying the Fortran-PL/I-PL/1-Simula-PL/P-PL/M/CHILL heritage popular in the 1970s*, which itself mimicked mathematics' usage of parentheses around each matrix. Because there was no way to represent mathematics' subscripts as the notation for indexing, the next best punctuation for matrix/vector indexing was borrowed: parentheses.

* as opposed to the ALGOL58's, ALGOL60's, ALGOL68's, BCPL's, C's square brackets, so the big split was somewhere around 1957 for FORTRAN (and whichever predecessor languages influenced it) and 1958 for ALGOL58 (and whichever predecessor languages influenced it), as opposed to APL's ⍳ iota which uses neither parentheses nor square brackets to pull out an element since 1966

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Dec 2020 20:37:31 +0100

> [...] the key tactical advantage of Ada's usage parentheses for array indexing is to accomplish a switcheroo

Not substitute, but to provide whatever implementation necessary. In fact Ada is limited in terms of abstractions. There still exist things which cannot be implemented by user-defined subprograms. Ideally there should be none. Whatever syntax sugar, there should be always a possibility to back it by a user-provided primitive operation.

> But for people like DrPi who seek contractual assurance of FP-purity of (all?) invoked functions (and overt declaration of impurity of other functions),

If they are unsatisfied with the higher abstraction level of Ada, they can switch to lower-level languages where implementation details are exposed in syntax. The best we can do is to explain why such exposure is a bad idea.

[Conceptually Ada has nothing to do with FP and I sincerely hope it never will.]

> This 1970s faux pas [...] is merely some tarnish that an AdaNG would fix

This would be highly undesired. On the contrary impure array implementations are all OK to implement various heuristics and caching schemes on the container side. In fact, Ada moved in that direction already by providing crude user-defined array indexing. Clearly as hardware evolves towards parallel architectures with partitioned memory, low-level arrays will be less frequently exposed in interfaces. Comparing older and newer Ada code we can see that trend of moving away from plain arrays.

Furthermore, purity of implementation is not contract, per definition of. Purity is a non-functional requirement.

There is only few areas of interest for such:

1. Compile-time evaluation/initialization of static objects and constraints.
2. Optimization, especially in the cases of fine grained parallelism.

In any case there is no reason to reflect that in the syntax, whatsoever.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 14:11:59 -0800

> If they are unsatisfied with the higher abstraction level of Ada, they can switch to lower-level languages where implementation details are exposed in syntax.

No, Dmitry, that is where you are wrong. In this regard, Ada is the lower-level, grungier, cruder, uncouther programming language, closer to assembly language or ALGOL60. Languages that have a pure keyword (or equivalent elective designator for compile-time purity enforcement throughout a call-tree of subroutines) are the ones that are high-level, cleaner, more-sophisticated, more-refined programming languages, closer to the lofty heaven of mathematics. This is actually a sad commentary on software engineering as a professed practice that we cannot even agree which programming-language feature-sets are higher-level versus lower-level, grungier versus cleaner, cruder versus more sophisticated, and uncouther versus more refined.

There is no good reason for Ada to lack all of the mechanisms to support FP (other than historical happenstance, then being substantially frozen in a Steelman mindset without any follow-on Stainlessman (arguably Ada95's, Ada2005's, Ada2012's would-be set of requirements that they have incrementally grown into) then Silverman (arguably SPARK's would-be set of requirements that is an ever-closer-to-finished work-in-progress) then Iridiumman then Goldman then Palladiummand then Platinumman evermore sophisticated requirements for a best-practices programming language to live up to as humankind's understanding of programming, system engineering, software engineering, and mathematics advances over time).

> Furthermore, purity of implementation is not contract, per definition of. Purity is a non-functional requirement.

So is all of Ada's rich typing/subtypes. Ada is simply capable of expressing some categories of nonfunctional requirements of the design (e.g., rich typing) but not other more-modern categories of nonfunctional requirement (e.g., a pure keyword).

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 19 Dec 2020 13:51:35 -0800

> Reading the code can't tell you the writer's intentions.

That's what comments and design documents are for.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sat, 19 Dec 2020 14:20:52 -0800

> > Reading the code can't tell you the writer's intentions.

> That's what comments and design documents are for.

For decades, assembly-language programmers said the same thing about structured-programming feature-set as being representable in mere comments & design documents. For decades, C programmers said the same thing about Ada's and C++'s and now Rust's feature-sets as being representable in mere comments & design documents. Arguably, the entire history of programming from Fortran (1957) and ALGOL (1958) forward is to encode the designer's intentions in source code that is vetted by a compiler instead of merely letting comments and design documents bit-rot as the declarative & imperative source code marches onward in the flow of time during initial greenfield completion (after all the “then a miracle occurs” on the blackboard sketches become rubber meeting road) and then during maintenance (as the design incrementally changes).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 20 Dec 2020 09:47:50 +0100

> No, Dmitry, that is where you are wrong. In this regard, Ada is the lower-level, grungier, cruder, uncouther programming language, closer to assembly language or ALGOL60.

Then we disagree on the definition of higher level. Mine is the level of abstraction away from calculus toward the problem space entities.

[...]

> So is all of Ada's rich typing/subtypes. Ada is simply capable of expressing some categories of nonfunctional requirements of the design (e.g., rich typing) but not other more-modern categories of nonfunctional requirement (e.g., a pure keyword).

The abstract datatype (in its original sense, rather than as abstract type in Ada) is meant to be a part of abstraction expressing the problem space. Purity of whatever implementation has nothing to do with the problem space. It is a design artifact.

Moreover, from the standpoint of programming paradigm, the whole procedural decomposition is lower level than OO decomposition done in terms of types and sets of types.

FP sits firmly in the procedural world. Even ignoring all fundamental flaws of FP concept, you will find no interest in FP from my side.

From: DrPi <314@drpi.fr>
Date: Sun, 20 Dec 2020 15:10:47 +0100

>> Reading the code can't tell you the writer's intentions.

> That's what comments and design documents are for.

A good IDE with code analysis showing you object declaration/use is very useful. Especially when comments are out of sync with the code.

I'm surprised that no modern tool/language allows the programmer to embed a "complete" documentation in source files. I'm not talking about comments formatted to suit a specific tool convention, like Python or Perl doc-strings. I'm talking about embedding schematics, drawings, bitmaps, mathematical equations, etc directly in the source code. Or maybe the reverse: embed source code in standard document. Like javascript in SVG files. Why not a .odt file with code sections? Ok, a specific file format would be better. Of course, the editor should be specific. No more a simple text editor.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Sun, 20 Dec 2020 08:53:36 -0800

> Then we disagree on the definition of higher level. Mine is the level of abstraction away from calculus toward the problem space entities.

Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) is clearly not abstraction. It is mere self-imposed blindness, ignoring the purity-enforcement topic altogether. Assembly language and Ada have the same inability to overtly express and enforce a declaration of FP-purity. Other languages have a pure keyword or equivalent for subroutines (i.e., functions, procedures, lambdas, coroutines, generators) to overtly express compile-time-enforced purity of the subroutine not making modifications to any data outside of its parameter data and callstack-based transient data. Clearly when a programming language (i.e., Ada) and assembly language share the same lack of feature, they are the more-primitive. Clearly when other pure-keyword-equipped programming languages can facilitate & enforce a higher civilization to capture the finer points of a

mathematical description of the problem domain via a rule-declaration & compile-time enforcement that assembly language lacks, they are higher-order and less primitive. There is no valid definition of "higher-order programming language" that permits assembly language's lack of a pure keyword (or equivalent purity-enforcement mechanism) to be a higher-order language than, say, Scala with a pure keyword. Dmitry, your line of reasoning here of what constitutes a higher-order language is preposterous!

From: Stéphane Rivière <stef@genesix.fr>
Date: Tue, 22 Dec 2020 11:05:10 +0100

> Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to .../...

Thanks for your message. It makes my day. I'm not fluent as you in english, nor in Ada concepts (I just use it with joy), but let me express my admiration for assertions such as:

> Assembly language and Ada have the same inability to overtly express and enforce a declaration of FP-purity.

Although this thought also plunges me into an abyss of reflection:

> Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) is clearly not abstraction.

There remains a mystery.

Why does your message remind me of this scene from another genius, Stanley Kubrick?
https://www.youtube.com/watch?v=iAHJCPoWCC8

No need to answer me, I don't have your skills to debate it. Just be assured that this post is not mocking and more expressing amazement.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 18:58:51 -0600

>Ada's inexpressiveness of imprecision of vagueness of misrepresenting design intent in this regard (of inability to compile-time enforce purity of subroutines) ...

Which Ada? Ada 202x has Global aspects specifically for this purpose, and they are compile-time enforced. Methinks are you simply looking to troll Ada rather than any serious intent.

There's no implementation of Global yet, sadly. Hopefully coming soon.

From: Andreas Zuercher <zuercher_andreas@outlook.com>
Date: Mon, 21 Dec 2020 18:39:45 -0800

> Ada 202x has Global aspects specifically for this purpose, and they are compile-time enforced.

This is very good news. I will need to investigate those AIs further. I take it from your wording that Global aspects are a general mechanism that a codebase could use to implement e.g. the purity check that FP seeks. If a general mechanism, it will be interesting to foresee what other categories of axioms can be enforced/assured beyond purity. Btw, I botched my example of extant programming languages in a prior comment that has a purity check on a call tree. D has it currently, but it has been proposed but not yet incorporated into Scala.

> Methinks are you simply looking to troll Ada rather than any serious intent.

No, absolutely not, at least not in the pejorative [sense] that your wording implies. As a system-engineer •critic• of finding the flaws in the system at large, I am always performing gap analysis on current Ada versus desired state of a universal programming language, using a technique not unlike FMEA. At some level you are coincidentally correct: I am negatively disappointed with Ada as much as C++ as much as Scala as much as D as much as Kotlin as much as Swift as much as C# as much as OCaml, but in different ways and to different degrees for each language.

For example, I admire so many portions of Ada, especially its declarative rich typing expressivity and its 35-year lead in accomplishing much of what C++20 will finally get with their oft-pursued concept feature. Conversely, it is sad that few people realize that Ada has had much of the new whizbang C++20 concept feature for 35 years.

It is as if Ada is a mostly superior product whose salesmen don't consummate as many sales contracts as they ought. It is useful to study in depth precisely why the superior product partially fails to achieve its potential glory.

One of the most interesting successes of Ada is that its user community seems to have fairly consistently utilized the vast majority of the features of the language on a regular basis. Despite C++'s perceived popularity by comparison, each C++ codebase utilizes 10% of C++, but worse it is a different 10% of C++ utilized for each different codebase with vast rivalry between codebases regarding which portions of C++ are God's gift to humankind and which portions of C++ are uncouth. Hence, C++'s perceived popularity is more of a mirage than it first appears because there is no one C++ that is popular, but rather a hundred subsets of C++, 75 of which are intensely unpopular to each of the others and 24 of which are eye-rollingly barely tolerable to each of the others.

As no small achievement, Ada achieves Scott McNealy's "all the wood behind one arrow" vastly more than, say, C++'s or D's everything-and-the-kitchen-sink pandering to me-too-isms. Scala/JVM, Scala/Native, Scala/OO, and Scala/FP are constantly in a multi-way tug-of-war of sorts (actually 2 orthogonal tugs-of-wars at 2 different ontological levels) that again isn't "all the wood behind one arrow" that Ada better achieves than Scala (so far).

> There's no implementation of Global yet, sadly. Hopefully coming soon.

It will be interesting to see the furthest push-the-limits extent of applicability of Global aspects.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Sun, 20 Dec 2020 13:59:20 -0800

I've never found any of the arguments in favor of using parentheses for array indexing convincing, and I've never liked the way Ada does it. But of course the decision was made in the early 1980s, and it can't be changed now.

At least part of the reason was that Ada needed to be used on systems that didn't have '[' and ']' in their character sets. I don't know to what extent that necessity has been used as an after the fact rationalization.

Function calls and array indexing can be substituted for one another in *some* circumstances, but not in all. But they really are very different things. A function call executes user-written code, and may have side effects; an array indexing expression refers to an object. An array indexing expression can appear on the LHS of an assignment; a function call can't.

If Ada had originally used '[' and ']' for array indexing, I doubt that anyone would be complaining that it would have been better to use '(' and ')' (other than some Fortran programmers, I suppose).

Why not use parentheses for record components, Object(Component) rather than Object.Component Doesn't the same argument apply?

> There are proposals to allow [] as well as (), mainly to increase familiarity for new Ada users.

Ick. The only thing more confusing than using () for array indexing would be allowing either () or [] at the programmer's whim. (Well, not the only thing; I'm sure I could come up with something even worse.)

> Parameterless procedures are much more common. Writing

> Frobnicate_Widget();

> is longer than

> Frobnicate_Widget;

> and seems to have no advantages over the shorter form.

I wouldn't have expected the designers of Ada to be concerned about saving two characters.

I see your point about procedure calls. A statement consisting of an identifier followed by a semicolon can only be a procedure call (I think), so there's no ambiguity. My mild dislike for the function call syntax is that it needlessly treats the zero-parameter case as special.

There could also be some potential ambiguities, though I'm not aware of any actual ambiguous cases in Ada. In some languages, the name of a function not followed by parentheses refers to the function itself (or its address) and does not call it. I can easily imagine an attribute for which Func'Attribute could sensibly refer either to the function Func itself or to the value returned by calling it.

Again, if Ada 83 had required empty parentheses on parameterless procedure and function calls, I'm skeptical that anyone would now be arguing that it was a bad decision.

And again, it would be impossible to change it without breaking existing code.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 21 Dec 2020 09:08:30 +0100

> Function calls and array indexing can be substituted for one another in *some* circumstances, but not it all.

IMO the only circumstances violating this substitutability are language design bugs and deficiencies:

- Passing array elements in in-out mode

- Assigning array elements

- Multidimensional indices

- Slices

all these must be substitutable with user-defined subprograms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:04:43 -0600

> Function calls and array indexing can be substituted for one another in *some* circumstances, but not it all.

This is false in modern languages with user-defined indexing (Ada and C++ included), since what looks like array indexing can actually be implemented with a function call.

Not having variable returning functions is a flaw in Ada, IMHO. These days, I think there are still too many special cases in Ada. If I was starting today, () would be a function call, and . would be selection/dereferencing, and there would not be anything else (which means getting rid of type conversions, array indexing and slicing, and anything else I've forgotten about). Compilers are smart enough to generate better code when they know something about the function involved (including if it is that of a predefined container). Doing that would allow overloading to be more general and to allow for the complication of variable returning functions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 22 Dec 2020 09:00:14 +0100

> If I was starting today, () would be a function call, and . would be selection/dereferencing, and there would not be anything else

But you cannot get rid of X(...) syntax, where X is an object. It is not only indexing, e.g. in declarations:

```
X : T (Y);
```

Then what is wrong with indexing? It should simply apply to all types [from some predefined class]:

```
X (...) ::= CALL (<index-operation>, X, ...)
(...) ::= CALL (<aggregate-operation>, ...)
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 22 Dec 2020 19:23:51 -0600

> Then what is wrong with indexing?

Nothing is "wrong" with it, it is just redundant. As others have noted here, both indexes and function calls represent a mapping. What's the point of having two ways to represent a mapping? In an Ada-like language, there's no syntax nor semantic difference.

Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for interesting stuff (static analysis, parallel execution, etc.).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Dec 2020 09:59:46 +0100

>> But you cannot get rid of X(...) syntax, where X is an object.

> That's a prefixed view, of course. No one would want to get rid of that.

Hmm, where is the operation? A prefixed view is

```
<expression>.<operation>(...)
```

Indexing is

```
<expression>(...)
```

In particular:

```
"abc"(1)
```

>> It is not only indexing, e.g. in declarations:

```
>>    X : T (Y);
```

> That's not an expression and is not resolved (that is, there is no possible overloading).

I see no fundamental difference between "first-class" expressions and type-expressions.

>> Then what is wrong with indexing?

> Nothing is "wrong" with it, it is just redundant. As others have noted here, both indexes and function calls represent a mapping. What's the point of having two ways to represent a mapping? In an Ada-like language, there's no syntax nor semantic difference.

Both are mappings, but unless you make functions first-class citizens there exist language level differences between a function and a container object.

> Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for interesting stuff (static analysis, parallel execution, etc.).

Yes, but I would rather keep all this stuff in the language making it overridable primitive operations.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 23 Dec 2020 22:06:03 -0600

> Hmm, where is the operation? A prefixed view is

```
>    <expression>.<operation>(...)
```

> Indexing is

```
>    <expression>(...)
```

I neglected to mention that what Ada calls objects are also function calls in this proposed generalization. (Much like enumeration literals are in Ada.) So for static semantics (that is, compile-time), pretty much everything is a function call. This gets rid of the anomalies associated with constants (which don't overload and thus hide more than a parameterless function - which is otherwise the same thing); combined with variable-returning functions, everything is overloadable and treated the same in expressions. Almost no special cases (operators still require some special casing, but we can make them always visible which would eliminate more issues).

Clearly a compiler for this language (which can't be Ada, unfortunately, way too incompatible) would special-case some kinds of built-in functions for things like objects and indexing. But that doesn't need to hair up the semantic model, just the implementations.

>> Ada (and most other languages) are full of redundant stuff. Simplify the basics and then one has more room for

interesting stuff (static analysis, parallel execution, etc.).

> Yes, but I would rather keep all this stuff in the language making it overridable primitive operations.

Yeah, you don't plan to formally describe nor implement this language, so you don't really care about how complex it gets. :-) Well, at least not until performance suffers. Ada is reaching the limit of what can be done without substantial incompatibility. If we're going to allow that, we need to start with a cleaner base, and part of that is getting rid of redundancies.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Dec 2020 10:37:10 +0100

> I neglected to mention that what Ada calls objects are also function calls in this proposed generalization.

Well, you must stop the recursion somewhere. It is fine to treat access to objects as calls, e.g. to getter/setter, or to indexation, or to dereferencing, but you must finish at some point with something spelled as a call to a subprogram. In the case of a subprogram call you are already there. With "objects" you need a few hops to get there.

[...]

> Ada is reaching the limit of what can be done without substantial incompatibility. If we're going to allow that, we need to start with a cleaner base, and part of that is getting rid of redundancies.

We see that differently. So far new features were added on top which naturally leads to the mess we observe. The problem is lack of generalization not inconsistency. If the new Ada cannot express the old messy, but consistent Ada, then this new Ada is not general enough and it will arrive at the same amount of mess sooner or later.

Getting Integers from Strings

From: John Perry <john.perry@usm.edu>
Subject: Help parsing the language manual on Get'ing integers from Strings
Date: Sun, 20 Dec 2020 16:11:43 -0800
Newsgroups: comp.lang.ada

Sorry if the subject is unclear. I recently tried to use

```
Get(S, Value, Last);
```

...in a program where Value was a Natural and S has the value "29: 116 82 | 119 24". GNAT gave me a Data_Error.

I don't understand why. [...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Dec 2020 09:44:30 +0200

[...]

It seems that the Get procedure understands ':' as a base indicator, as in

"12#44#" works, Value = 52, Last = 6.

"12#44" fails with Data_Error.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 21 Dec 2020 08:57:36 +0100

[...]

Colon: is a replacement character for # (see allowable replacements of characters). So it might think of 29: 116 as a malformed base-29 number with wrong base and missing closing:.

[...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Dec 2020 10:06:47 +0200

I see, an "obsolescent feature" in RM J.2. I learn something new every day (I hope).

Ok, so no bug.

From: Jeffrey R. Carter
Date: Mon, 21 Dec 2020 10:40:17 +0100

> I see, an "obsolescent feature" in RM J.2.

Yes. I never worked with a system that required such substitutions, even in 1984 when it was not an obsolescent feature, but as we can see, it's important to be aware of them.

These days they are sometimes used for obfuscation.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:11:32 -0600

> Yes. I never worked with a system that required such substitutions, even in 1984 when it was not an obsolescent feature, but as we can see, it's important to be aware of them.

I believe that restriction had to do with certain keypunches. But hardly anyone used keypunches even in 1981. (The Unisaur computer that our CS compiler-construction class used still had a few keypunches, but they had mostly transitioned to terminals by that time. I think that was the last class to use the Unisaur; they just had installed some VAX 780s for research and they soon got some for student use as well. My first few programming classes at UW used the Unisaurs keypunches.) I think that requirement was obsolete by the time Ada was completed (it probably wasn't when the Ada design was started).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Dec 2020 19:19:44 -0600

> Perhaps RM-A.10.8(8) should be clarified/corrected.

For what it's worth, we once tried to do that, but couldn't come to an agreement on precisely what to change the wording to. As a change is not critical, we didn't make one. The ACATS has long had tests in this area that require something subtly different than the wording requires, and it didn't make any sense to change them (since presumably all implementers are passing them, rather than strictly following the RM wording).

In any case, the ":" replacement trips up people from time-to-time, as pretty much no one remembers it. I recall we had to change some piece of new syntax because the possibility of a colon in a number made it ambiguous.

On the Future of the Distributed Systems Annex

From: Rod Kay <rodakay5@gmail.com>
Subject: 2dsa | !2dsa?
Date: Tue, 22 Dec 2020 12:00:48 -0800
Newsgroups: comp.lang.ada

I've heard that the Distributed Systems Annex (DSA) may be dropped from the Ada standard soon. Can anyone confirm this?

I've been using the PolyORB implementation of DSA for some time and find it very useful. The way in which it abstracts away socket 'plumbing' details makes for very simple/understandable comms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 22 Dec 2020 19:32:37 -0600

> I've heard that the Distributed Systems Annex (DSA) may be dropped from the Ada standard soon. Can anyone confirm this?

Annex E remains in the proposed Ada 202x standard.

Compiler support, of course, is up to vendors. Dunno if anyone is still supporting it.

> I've been using the PolyORB implementation of DSA for some time and find it very useful. The way in which it abstracts away socket 'plumbing' details makes for very simple/understandable comms.

That was the promise, not sure it ever really was realized. Since the Annex was weakened enough that third-party support isn't really possible anymore (necessary to allow it to be used with current middleware), it's really a vendor-specific thing these days.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Dec 2020 09:44:26 +0100

> Compiler support, of course, is up to vendors. Dunno if anyone is still supporting it.

It should be moved to the user level. As specified in the Annex there seems no obvious way to provide a user-defined transport for DSA, and there seems no way to have different implementations of DSA in the same program.

[...]

> [...] it's really a vendor-specific thing these days.

Yes, I always wished to include DSA support based on various communication protocols I have implemented in Ada, rather than plain sockets. E.g. I have a ready-to-go DSA implementation for interprocess communication over shared memory, but no idea how to make GNAT aware of it. Or AQMP and ASN.1 look like a straightforward candidate as a DSA transport as they have detailed type description systems to map Ada objects.

From: Maxim Reznik <reznikmm@gmail.com>
Date: Thu, 24 Dec 2020 04:02:24 -0800

I forked an older (Garlic) GNAT DSA implementation and found it quite hackable. :)

My idea is to implement a WebSocket/WebRTC transport and compile it by GNAT-LLVM to WebAssembly to have distributed Ada applications in a browser. I have a working proof of concept demo already :)

https://github.com/reznikmm/garlic/tree/web_socket

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Dec 2020 14:30:54 +0100

> I forked an older (Garlic) GNAT DSA implementation and found it quite hackable. :)

My question is how to proceed without GLADE/Garlic etc. I have DSA implemented, e.g. System.RPC. I need GNAT to accept it as such.

In a more distant perspective I need a work-around of stream attributes. They are non-portable, so there must be an option to replace them for DSA and/or provide a non-stream parameter marshaling when the transport is a higher-level protocol, e.g. CANopen, EtherCAT, ASN.1, AMQP etc. For these you must map Ada types into the types defined by the protocol. Without this DSA is pretty much pointless.

From: Rod Kay <rodakay5@gmail.com>
Date: Sun, 27 Dec 2020 11:34:10 -0800

Is it likely that the ARG might address the aforementioned issues?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Dec 2020 17:41:39 -0600

>Is it likely that the ARG might address the aforementioned issues?

As of now, it doesn't appear that there would be any point. Annex E is an optional annex, and so far as we're aware, no compiler vendor has any plans for increasing support for that annex. So the ARG could change the annex but it seems unlikely that any changes would make it into implementations. (We've been told not to expect even the implementation of bugs fixes included in Ada 202x, even from the vendor that originally requested the bug fixes.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 29 Dec 2020 15:56:35 +0100

>> Is it likely that the ARG might address the aforementioned issues?

> As of now, it doesn't appear that there would be any point.

Why should there be any vendor support in the first place? Why not to redefine it as a set of abstract tagged types allowing custom user implementations like storage pools and streams do?

The idea of having an IDL, statically assigned partitions, linking everything together before start is not the way the distributed systems are designed and work today. CORBA died for a reason.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 29 Dec 2020 16:51:19 +0100

> Would the compiler still need any support for this or would it just be a set of interfaces at library level?

Yes, because the idea is to have remote objects and remote calls looking exactly the same as local objects and local calls.

So the compiler must translate a call to an RPC to a call to some user primitive operation like System.RPC does. The operation would have a controlling parameter "connection" or "remote partition". The actual input values of the original call must be marshaled, e.g. as an output stream. The output values and the result will be returned via an input stream and deserialized from there into the actual parameters/result or else into a remote exception occurrence to re-raise locally if that was the outcome.

Here lie a lot of problems. First is non-portability of stream attributes. Second is lack of support for bulky transfers and multiplexing. It is highly desirable that the output stream could be written in chunks as well as reading the input stream. E.g. if you pass large objects or if you want to multiplex RPCs made from different tasks rather than interlock them (which for synchronous RPC would result in catastrophic performance).

The current Annex E is very crude to allow efficient, low-latency, real-time implementations.

P.S. If Ada supported delegation, introspection and getter/setter interface, then, probably, all remote call/objects stuff could be made at the library level. But for now, compiler magic is needed.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 31 Dec 2020 17:43:39 -0600

> Why should there be any vendor support in the first place? Why not to redefine it as a set of abstract tagged types allowing custom user implementations like storage pools and streams do?

Marshalling/unmarshalling surely require vendor support, and there has to be a standard interface for the marshalling stuff to talk to. That to me was always the value of Annex E. My understanding is that there is not much interest in doing any work at all, even to correct the mistakes in the existing definitions.

In any case, Storage_Pools and Streams are some of the most expensive features of Ada to support. That's not a model for "lightweight" support of anything.

My advice would be to talk to your vendor if you feel strongly about this sort of support.

Easiest Way to Use Regular Expressions?

From: reinert <reinkor@gmail.com>
Subject: Easiest way to use redular expressions?
Date: Sun, 27 Dec 2020 00:20:11 -0800
Newsgroups: comp.lang.ada

I made the following hack to match a string with a regular expression (using a named pipe and grep under linux):

[Omitted example of spawning a process and capturing the output. —arm]

OK, I assume it somehow breaks the philosophy on Ada and security/reliability. Could someone therefore show a better and more simple way to do this? gnat.expect?

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 27 Dec 2020 09:36:49 +0100

AdaControl uses Gnat.Regpat, and is quite happy with it...

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 27 Dec 2020 03:14:47 -0800

> AdaControl uses Gnat.Regpat, and is quite happy with it...

GNAT.Regpat is a package I wrote 18 years ago or so (time flies..), basically manually translating C code from the Perl implementation of regular expressions. Nowadays, I think it would be better to write a small binding to the pcre library (which has quite a simple API, so the binding should not be too hard). This will provide much better performance, support for unicode, and a host of regexp features that are not supported by GNAT.Regpat.

Never did that while I was working for AdaCore because we would have ended up with too many regexp packages (there is also GNAT.Regexp, which is very efficient but limited in features because it is based on a definite state machine).

I think libpcre might even be distributed with gcc nowadays, although I did not double-check so might be wrong.

This binding would be a nice small project for someone who wants to get started with writing Ada bindings

From: Maxim Reznik <reznikmm@gmail.com>
Date: Mon, 28 Dec 2020 05:58:06 -0800

The Matreshka library has rather advanced regexp engine with full Unicode support

https://forge.ada-ru.org/matreshka/wiki/League/Regexp

From: Jeffrey R. Carter
Date: Mon, 28 Dec 2020 22:07:24 +0100

You can use PragmARC.Matching.Regular_Expression or its instantiation for Character and String, PragmARC.Matching.Character_Regular_Expression

https://github.com/jrcarter/PragmARC/tree/Ada-12

Renames Usage

From: DrPi <314@drpi.fr>
Subject: renames usage
Date: Thu, 31 Dec 2020 12:48:25 +0100
Newsgroups: comp.lang.ada

One can read here https://github.com/AdaCore/svd2ada/blob/master/src/descriptors-field.adb#L83 this line:

```
Tag   : String renames Elements.Get_Tag_Name (Child);
```

Is it equivalent to the following line?

```
Tag: String:= Elements.Get_Tag_Name (Child);
```

From: John Perry <john.perry@usm.edu>
Date: Thu, 31 Dec 2020 04:10:21 -0800

No. Assignment copies the object, and changes to the copy don't affect the original, while renaming obtains a reference to the object. [...]

From: Gautier write-only address <gautier_niouzes@hotmail.com>
Date: Thu, 31 Dec 2020 04:33:30 -0800

> Tag : String renames Elements.Get_Tag_Name (Child);

> Is it equivalent to the following line ?

> Tag : String := Elements.Get_Tag_Name (Child);

Since the temporary object containing the result of the call "Elements.Get_Tag_Name (Child)" is not accessible anywhere else, the effect is the same.

But, perhaps in some implementations, the "renames" accesses that temporary object, which means the memory containing it must not be freed until Tag is out of scope. Perhaps it is even required. Any compiler specialist here?

From: Jeffrey R. Carter
Date: Thu, 31 Dec 2020 15:49:04 +0100

> Tag : String renames Elements.Get_Tag_Name (Child);

> Is it equivalent to the following line ?

> Tag : String := Elements.Get_Tag_Name (Child);

No. A function result is a constant, so the 1st version gives you a constant. The second gives you a variable with the same value.

From: DrPi <314@drpi.fr>
Date: Thu, 31 Dec 2020 16:55:34 +0100

> No. A function result is a constant, so the 1st version gives you a constant. The second gives you a variable with the same value.

Good to know.

What disturbed me was the function call associated with "renames".

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 31 Dec 2020 19:48:43 +0100

> What disturbed me was the function call associated with "renames".

Renaming a call to a function does not rename it in some functional-programming manner. It renames only the result of.

So if you do

```
X : Float renames Random (Seed);
Y : array (1..10) of Float := (others => X);
```

That would not give you ten pseudo-random numbers. But this will:

```
Z : array (1..10) of Float := (others => Random (Seed));
```

Obituary

Tragic News about Vinzent Hoefler

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Tragic news about Vinzent Hoefler
Date: Wed, 28 Oct 2020 11:09:09 -0000
Newsgroups: comp.lang.ada

Dear all,

Many of you may know Vinzent Höfler.

I am sad to pass the most tragic news that Vinzent died last Wednesday 21 October...

Below is the message Vinzent's wife Katja Saranen asked me earlier today to share with the Ada community.

He was active in various forums and newsgroups as Vinzent aka "Jellix" aka "JeLlyFish.software@gmx.net" aka "ada.rocks@jlfencey.com" aka "vinzent@heisenbug.eu". He worked on professional as well as open-source Ada projects, was a member and participated in events of ACM SIGAda, Ada-Europe and Ada-Belgium, and helped with several recent Ada DevRooms at FOSDEM events.

I first met Vinzent what seems an eternity ago at the SIGAda 2002 conference in Houston. Our paths crossed many times since, until five years ago he became an "Ada" colleague at Eurocontrol.

I will miss Vinzent, as a colleague, as a like-minded spirit on various issues, and most of all as an intelligent human being. I will miss our interesting discussions: we had too few...

Dirk

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Message from Katja Saranen, Wed Oct 28 2020:

**

With the deepest sorrow I have to share with you a devastating tragedy.

Our beloved Vinzent has left this world, he is not with us anymore.

Vinzent "Jellix" Saranen (Höfler, Fritzsche)

09.01.1974 - 21.10.2020

Unspeakable loss for so many. A father, son, brother, grandfather, husband, friend, colleague and much more.

The love of my life. My soulmate. My person. My husband. My safe place. Incredible, wonderful, beautiful, weird, intelligent, talented. So special in so many ways.

We were supposed to grow old together and move to wilderness. I was not supposed to outlive you. I was not supposed to face the world without you. I don't know yet how am I even expected to keep going without you on my side.

This is not a farewell. You're not gone. Love is not any less. You're always with me until we meet again. Love you, forever.

starlingc/katja

"Death is that state in which one exists only in the memory of others, which is why it is not an end. No goodbyes. Just good memories. Hailing frequencies closed, sir."

(Star Trek TNG; Tasha Yar)

There will be no funeral or grave. He has been cremated yesterday and next summer I will take his ashes to the place where he was happiest and where he wanted to go to grow old. For a place to remember him, you can go to nature anywhere and you'll always be close. Memorial(s) will be planned at later time, I am not able to now.

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 28 Oct 2020 07:24:32 -0700

Tragic news indeed.

Sorry to see him go.

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Wed, 28 Oct 2020 11:35:56 -0700

> Tragic news indeed.

> Sorry to see him go.

Rest in peace. Sincere condolences.