ADA USER JOURNAL

Volume 43
Number 2
June 2022

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,

It is a recurrent topic that the several GNAT versions with differing licensing conditions are [not] a hindrance to Ada adoption. Well, AdaCore has announced that, moving forward, their open source oriented compiler offering will be unique and based on the FSF source tree, hence with similar licensing as other GCC languages [1]. Let us hope this puts to rest the FUD of the past. No doubt, a remarkable piece of news and a valiant bet on the future of Ada with the backing of a healthy open source community.

Likewise on the open source front, the HAC compiler continues making steady progress [2], with several versions released in close succession, each one bringing more Ada features into its supported subset.

This issue has seen a livelier than usual section on Ada and other languages. I always find interesting the examination of the interrelations between languages, and how Ada fits in the past, present and future of programming languages. Will Ada ever regain a spot in the top-10 more popular languages [3]? Perhaps we should devise our own ranking using Metrics that Really Matter™ ;-)

Sincerely,
Alejandro R. Mosteo.

[1] "A New Era for Ada/SPARK Open Source Community", in Ada-related Tools.

[2] "HAC v.0.2", in Ada-related Tools.

[3] "When Ada Was the Most Popular Language", in Ada and Other Languages.

Ada-related Events

Ada Monthly Meeting Proposal

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: Ada Monthly Meeting proposal
Date: Tue, 26 Apr 2022 21:59:32 +0200
Newsgroups: comp.lang.ada

Ada Monthly meeting

A lot of programming languages and libraries have meetings/meetups which allow the community to come together and have a chat, share ideas, proposals and better utilise and prioritise resources. I would like to propose such a thing for Ada. Below is the rationale and some ideas and issues.

Motivation

The Ada community does not have many members when compared to other more-well-known communities. However, there is still some interest in having such type of meetings. This was recently made clear after some people pointed out they would like to have such a thing in the Ada channel over at Gitter. Personally, I have been playing with such an idea and that is what motivated me to volunteer to drive the FOSDEM Ada devroom. I would like the Ada community to be more well-known and to have the same "resources" as other communities have. Meetups are a great way to have fun while discussing what we like.

In meetups, generally speaking, users and developers can have the opportunity to come together, discuss topics, organise resources and help each other. Gray-beards can help those who still have colour in their hair; people with different sets of skills can propose solutions to problems that one may not have thought about; authors can present their work or improvements, etc.

I would like to use the Fortran community as an example of what meetups can be used for. Here is their April monthly-meetup [1]. Their meetups are very focused on the language and "core" tooling... it is quite formal, which may not be what I had in mind, but we will see.

Who is this meant for?

Everybody who is interested.

I would love to see some participation of the "Industrial users". But I understand that a lot of people see Ada (and many other things) as a tool that brings food to the table, nothing more. So I would not expect much participation from this group.

Newbees and beginners are also more than welcome. They could see what people are doing and ask questions that are better answered in real-time by a person, instead of a Stackoverflow for example.

Though, I must be honest, it is mostly intended for people who are interested in the Ada environment and open side of things. This is due to the nature of an open discussion and building a community. I have to be clear and state that I am biased towards the libre community, so feel free to point out any unfairness.

How would it work? What would it be like?

THIS IS JUST A PROPOSAL, SO TAKE THIS AS SUCH.

I thought about having a Jitsi room (libre conference system that runs on your browser, same one used in FOSDEM) [2] where people can just join and take part of the meetup. Jitsi allows for moderation too, so that speakers can talk without getting interrupted and it has a built in chat too.

So, what could be discussed? Here is a short list of ideas that I have:

- Monthly news: new releases, milestones, etc.
- Presentations: attendees may want to present their work or do a demonstration. They may also want to have a discussion about a specific topic (for example, the use of Ada 2022 features).

- General libre software coordination: improvements to tools, feedback, questions, past goals discussion, etc.
- General Q&A related to Ada and open to everybody.
- Finally, a beer.

I think this could take place between 30 min to 2 hours, depending on the load of that day. Presentations would obviously be much more casual and easy when compared to an actual conference.

Potential issues

1. Not enough interest.
2. Timezones! Users are mostly concentrated in Asia-Pacific/EU/USA, which makes coordination an absolute pain. A compromise could be found, or a different schedule each month in such a way that everybody benefits (and gets screwed) equally.
3. Organisation: there needs to be a main organiser and a second in command-
4. There also needs to be a medium in which to spread the word. C.L.A is a good starting point, but may not reach the wider community. It could be announced everywhere every month, but that is a tedious task.

Feedback

I have probably said enough, even if not everything has been said. So I would like to ask for your feedback and specially know if you would be interested.

Thank you for your time,
Fer

References

[1] https://invidious-us.kavin.rocks/watch?v=8-_ll4f0gN8

[2] https://meet.jit.si/

From: Maxim Reznik <reznikmm@gmail.com>
Date: Wed, 27 Apr 2022 03:34:13 -0700

I would give it a try!

From: Anton F. <imantonmeep@gmail.com>
Date: Wed, 27 Apr 2022 05:57:49 -0700

I would participate, this is a great idea!

From: Yossep Binyoum <yossep237@gmail.com>
Date: Thu, 28 Apr 2022 13:41:03 -0700

From Senegal, I totally agree with you. I give it a try

From: Stéphane Rivière <stef@genesix.org>
Date: Fri, 29 Apr 2022 16:38:39 +0200

Great idea!

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 29 Apr 2022 11:29:47 -0700

> * Ada Monthly meeting

Sounds interesting. I maintain Emacs Ada mode; this might be a good forum to get less formal feedback than the ada-mode mailing list provides, and to hear what other IDEs are doing for Ada.

> 1. There also needs to be a medium in which to spread the word. C.L.A is a good starting point, but may not reach the wider community. It could be announced everywhere every month, but that is a tedious task.

This sounds like a job for a bot; post the same announcement to a list of channels.

Anyone have a bot written in Ada?

From: Maxim Reznik <reznikmm@gmail.com>
Date: Fri, 29 Apr 2022 21:04:15 -0700

> Anyone have a bot written in Ada?

I have a bot in Ada for Telegram. It is a bridge between Telegram, Jabber, IRC channel. It also checks whether telegram newcomers are not bots.

I can write another one for announcements, but I'm not sure if announcing once a month is worth the time :)

From: Ada Forge <adaforge2022@gmail.com>
Date: Sat, 30 Apr 2022 06:57:40 -0700

> * Ada Monthly meeting

Nice initiative!

Take me into account ;-)

Some subjects I'd love to debate with connoisseurs:

* UTF8-Unicode-UCS: a lot of libraries are offering strings manipulation. State of the art? (Gnat extensions, GnatColl, Matreska, Gnoga, ...)
* OS system usage (as (system shell) scripts, in place of Perl, Python, …): GNAT extensions; Florist; GnatColl; SoWebIO; ...
* Windowing (2D) systems: future of GTK/Glade; Qt6/Qt Design Studio; GWindows; Apple new SwiftUI MV paradigm; wxWidgets; Tk/TCL
* How let anyone collaborate to AdaForge's new up-todate 2022 Ada resources gathered all over the internet ;-) Through GitHub?

Cheers, with a fresh Belgian Ada 10°
William

From: Rod Kay <rodakay5@gmail.com>
Date: Sun, 1 May 2022 02:33:47 +1000

> Nice initiative!

Agree. Count me in Fer :).

> Some subjects I'd love to debate with connoisseurs:

If by 'OS system usage' you mean using Ada to write shell-like scripts then you may be interested in aShell. It builds on Florist to allow Ada applets to more easily call and interact with OS commands.

The last release allowed OS commands to be called but only from a single Ada task. Atm, work is being done on supporting task safe commands (via a spawn manager).

The next release will contain the task safe commands and be Alire enabled, and should occur in the next month or so (Lady Ada willing).

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 30 Apr 2022 18:01:24 +0100

> Some subjects I'd love to debate with connoisseurs:

> * UTF8-Unicode-UCS: a lot of libraries are offering strings manipulation. State of the art? (Gnat extensions, GnatColl, Matreska, Gnoga, ...)

> * Windowing (2D) systems: future of GTK/Glade; Qt6/Qt Design Studio; GWindows; Apple new SwiftUI MV paradigm; wxWidgets; Tk/TCL

Oh, I suppose I'll have to attend given I have experience with those.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 3 May 2022 21:06:21 +0200

Thank you all for your answers :)

It seems that there is some interest to have a meeting from time to time. Other communication channels where this proposal was posted did have other people who liked the idea. For this reason, I would like to share some extra bits.

- I think the duration of such a monthly meeting could last for an hour, an hour and a half.
- Could take place monthly in a varying schedule to suite some people better than other depending on the month.
- The level of preparation is much much lower than FOSDEM or similar venues. This is more about building community than actually making this a serious thing.
- The structure could be the following:
  - < 5 mins to share important news and announcements.
  - < 30 mins reserved for already predefined talks/topics. More on this later.
  - < 15 min discussion topic. Maybe there is something that needs a few words, it's the topic of the day.
  - < 15 mins to let people share their work or improvements.
  - The main meeting ends here.

- Open questions and answers and general discussion/beer.
- The main section of the meeting (without Q&A and open discussion) could be recorded and uploaded to video hosting sites. I know a few people already reupload the videos from FOSDEM, so I could as them to do the same for us. This would allow us to keep a log of the meetings:)
- We could use Jitsi, a libre conference software. I know it has recording capabilities, but I think only for Youtube... :/ We will see whether Jitsi actually works or not...

If this works, I think we could start inviting people to share their work and reserve time for their presentations. That is what the second section of the proposed schedule is about. At the beginning, obviously, we will focus on making sure that the meetings work and see if there is enough recurring interest in them.

Regarding the actual planning. I will not make it for the month of May unless someone steps and helps a fair bit. On a personal note, I have a lot of work and it will just keep increasing so I cannot ensure that I will be able to pull something like this alone. FOSDEM was already a bit exhausting :P

I also want to see what you have to offer both in direct help or if you have projects that you want to talk about, presentations, etc. Some of you already commented on it, so I am happy.

What is your opinion about this? I would need feedback :)

Also, please, feel free to repost this to other social media. The more Ada users and people interested in Ada the better! If you want a contact, feel free to email me at "irvise(AT)irvise.xyz".

Bye now :D
Fer

CFP: ACM SIGAda HILT 2022 Workshop at ASE '22, October 14, 2022

From: Tucker Taft <tucker.taft@gmail.com>
Subject: CFP: ACM SIGAda HILT 2022 Workshop at ASE '22, October 14, 2022
Date: Thu, 12 May 2022 18:30:15 -0700
Newsgroups: comp.lang.ada

Please consider contributing to this workshop sponsored by ACM SIGAda: HILT-2022 - Supporting a Rigorous Approach to Software Development

This is the seventh in the HILT series of conferences and workshops focused on the use of High Integrity Language Technology to address challenging issues in the engineering of highly complex critical software systems.

High Integrity Language Technologies have been tackling the challenges of building efficient, safe, reliable software for decades. Critical software as a domain is quickly expanding beyond embedded real-time control applications to the increasing reliance on complex software for the basic functioning of businesses, governments, and society in general.

For its 2022 edition, HILT will be a workshop of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE'2022. The workshop will be held on October 14th 2022.

See ASE'2022 (https://conf.researchr.org/home/ase-2022) for details on the venue and registration.

Topics

HILT 2022 will focus on the increasing synergies between formal methods (theorem provers, SAT, SMT, etc.), advanced static analysis (model checking, abstract interpretation), software design and modeling, and safety-oriented languages. From separate fields of research, we now observe a stronger interconnection between formal methods, advanced analytics, modeling and design of software, and safety features in programming languages. Programming languages for safety-critical systems now routinely integrate theorem proving capabilities like C/ACSL or Ada/SPARK2014. Theorem provers such as Coq, Lean, or Isabelle have established themselves as a viable strategy to implement compilers or properly define the semantics of domain-specific languages. Tools for verifying modeling languages such as AADL, Lustre, and Simulink are becoming more widely available, and with the emergence of the Rust language and the release of Ada 2022, safety is rising to the top of concerns for critical systems developers.

The HILT'2022 workshop seeks to explore ways High Integrity Language Technologies leverage recent advances in practical formal methods and language design to deliver the next generation of safety-critical systems.

Call for Papers

This workshop is focused on the practical use of High Integrity languages, technologies, and methodologies that enable expedited design and development of software-intensive systems.

Key areas of interest include experience and research into:

Practical use of formal methods at industrial scale

IDE-support for formal methods

Model-level analysis tools for systems like SysML, AADL, Lustre, or Simulink

Continuous Integration and Deployment based on advanced static analysis tools

Safety-Oriented Programming Language features *Qualification of Language Tools for critical systems use

The workshop accepts either short abstracts (2-3 pages) for presentation, or full papers (up to 8 pages).

Submissions should conform, at time of submission, to the ACM Proceedings Template: https://www.acm.org/publications/proceedings-template.

The workshop proceedings will be published in the ACM Ada Letters. Authors of accepted papers will be invited to contribute to a special issue of the Springer Journal on Software and Tools for Technology Transfer (STTT).

Paper submission

Submit your paper through Easychair at https://easychair.org/conferences/?conf=hilt22

Important Dates

Submission Deadline: July, 1 2022

Notification to authors: August, 1 2022

Workshop Date: October 14th 2022.

From: Tucker Taft <tucker.taft@gmail.com>
Date: Thu, 12 May 2022 18:34:51 -0700

> Please consider contributing to this workshop sponsored by ACM SIGAda: HILT-2022 - Supporting a Rigorous Approach to Software Development

Website is:
https://conf.researchr.org/home/hilt-2022

Press Release - AEiC 2022, Ada-Europe Reliable Softw. Technol.

[The event took place during 14-17 June, so this announcement is for the record. —arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Press Release - AEiC 2022, Ada-Europe Reliable Softw. Technol.
Date: Sun, 12 Jun 2022 20:59:37 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FINAL Call for Participation

***UPDATED Program Summary ***

26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022)

14-17 June 2022, Ghent, Belgium

www.ada-europe.org/conference2022

Organized by Ada-Europe in cooperation with ACM SIGAda, SIGPLAN, SIGBED, the Ada Resource Association (ARA), and Ghent University

#AEiC2022 #AdaEurope #AdaProgramming

*** Final Program available on the conference web site ***

*** Add tutorials and/or a workshop to your conference registration ***

www.ada-europe.org/conference2022/tutorials.html

* Welcome Event on Tuesday evening *

Press release:

26th Ada-Europe Int'l Conference on Reliable Software Technologies.

International experts meet in Ghent.

Ghent, Belgium (12 June 2022) - Ada-Europe together with the University of Ghent, Belgium, organizes from 14 to 17 June 2022 the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022). The event is in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED) and on Programming Languages (SIGPLAN).

The Ada-Europe series of conferences has over the years become a leading international forum for providers, practitioners and researchers in reliable software technologies. These events highlight the increased relevance of Ada in general and in safety- and security-critical systems in particular, and provide a unique opportunity for interaction and collaboration between academics and industrial practitioners.

This year's conference offers 4 tutorials, a keynote and an invited presentation, a technical program of 7 sessions with refereed papers industrial, work-in-progress, and vendor presentations, a social program with exciting sightseeing, 2 workshops and a Birds-of-a-Feather session.

Four tutorials are scheduled on Tuesday, targeting different audiences:

- "Moving up to Ada 2022", by S. Tucker Taft, AdaCore, USA;
- "Numerics for the non-numerical analyst", by Jean-Pierre Rosen, Adalog, France;
- "The ALiRe Package Manager", by Fabien Chouteau, France, and Alejandro Mosteo, Spain;
- "The HAC Ada Compiler", by Gautier de Montmollin, Switzerland.

Vendors and organisations will be present in the networking area on Wednesday and Thursday include AdaCore, VECTOR, and Ada-Europe.

Two eminent speakers have been invited to deliver a talk on each of the core conference days:

- on Wed Jun 15, a spotlight talk (remote) by Anita Carleton, Software Engineering Institute, Carnegie Mellon University, USA, about "Envisioning the Future of Software Engineering";
- on Thu June 16, a keynote talk by Cristina (Crista) Lopes, School of Computer Sciences, University of California at Irvine, USA, who will present her study on "The Curious Case of Code Duplication in Github".

The technical program on Wednesday and Thursday presents 7 sessions with 9 journal-track refereed technical papers, 9 industrial, 12 work-in-progress, and 2 vendor presentations in sessions on: Uses of Ada, Real-Time Systems 1, Development Challenges, Advanced Systems, Special-Purpose Systems, Verification Challenges, Real-Time Systems 2.

On Friday the conference hosts for the 7th year the workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2022), as well as the International Workshop "AADL Unveiled by its Practitioners (ADEPT), and a Birds-of-a-feather (BoF) Meeting on the "Future of ASIS and Vendor Independent Tools".

Peer-reviewed papers have been submitted to a special issue of the Journal of Systems Architecture and are heading towards final acceptance as open-access publications. Industrial and work-in-progress presentations, together with tutorial abstracts, will be offered publication in the Ada User Journal, the quarterly magazine of Ada-Europe.

The social program includes for all tutorial and conference participants on Tuesday evening a Welcome Aperitif with beer tasting (sponsored by VECTOR) in the "Il Trovatore" lounge, a restored medieval cellar. On Wednesday evening, a private visit to the Gothic-style St Bavo's Cathedral and its artistic treasures including the world-famous Lam Gods altarpiece, followed by the Conference Banquet in the Abt, the only brasserie from the famous Orval Trappist beer brewery. And on Thursday evening a boat tour in the canals that encircle the medieval center of Ghent, followed by a conference dinner at the Carlos Quinto restaurant, a short walk across the heart of town from the boat pier.

The Best Presentation Award will be offered during the Closing session.

The full program is available on the conference web site.

Online registration is still possible.

Latest updates:

The 16-page "Final Program" is available at www.ada-europe.org/conference2022/docs/AEiC_2022_Final_Program.pdf.

Check out the tutorials in the PDF program, or in the schedule at www.ada-europe.org/conference2022/tutorials.html.

Registration is done on-line. For all details, select "Registration" at www.ada-europe.org/conference2022 or go directly to https://registration.ada-europe.org.

A printed Conference Booklet with abstracts of all technical papers

and industrial presentations will be included in every conference

handout, and is available at www.ada-europe.org/conference2022/docs/AEiC_2022_Booklet_of_Presentations.pdf.

AEiC 2022 is sponsored by Ada-Europe (www.ada-europe.org), AdaCore (www.adacore.com), and VECTOR (www.vector.com/int/en/products/products-a-z/software/vectorcast).

Help promote the conference by advertising it.

Recommended Twitter hashtags: #AEiC2022 #AdaEurope #AdaProgramming.

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2022 Publicity Chair

Dirk.Craeynest@cs.kuleuven.be

* 26th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2022)

* June 14-17, 2022, Ghent, Belgium * www.ada-europe.org/conference2022

(V6.1)

Ada/SPARK Crate of the Year 2022

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: Ada/SPARK Crate Of The Year is back!
Date: Tue, 28 Jun 2022 05:55:20 -0700
Newsgroups: comp.lang.ada

https://blog.adacore.com/announcing-the-2022-ada-spark-crate-of-the-year-award

[AdaCore offers 3 prizes of $2,000 each for the following categories: best overall

Ada crate, best crate written in SPARK and/or contributing to the SPARK ecosystem, and best Ada or SPARK crate for embedded software. Candidates can be submitted until the end of the year. —arm]

Ada-related Resources

[Delta counts are from May 9th to July 18th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 18 Jul 2022 14:53 CET
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_328 (+26) members [1]
- Reddit: 8_078 (+73) members [2]
- Stack Overflow: 2_238 (+26) questions [3]
- Libera.Chat: 75 (=) concurrent users [4]
- Gitter: 123 (+8) people [5]
- Telegram: 143 (+4) users [6]
- Twitter: 30 (=) tweeters [7]
  75 (+22) 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=Libera.Chat

[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: 18 Jul 2021 15:16 CET
To: Ada User Journal readership

Rosetta Code: 915 (+15) examples [1]
  39 (+1) developers [2]
GitHub: 763* (=) developers [3]
Sourceforge: 244 (-30) projects [4]
Open Hub: 214 (=) projects [5]
Alire: 260 (+17) crates [6]
Bitbucket: 87 (-1) repositories [7]
Codelabs: 53 (=) repositories [8]
AdaForge: 8 (=) repositories [9]

*This number is unreliable due to GitHub search limitations.

[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: 18 Jul 2021 15:51 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. —arm]

- TIOBE Index: 30 (-3) 0.38% (-0.08%) [1]
- PYPL Index: 17 (=) 0.86% (+0.05%) [2]
- IEEE Spectrum (general): 31 (=) Score: 38.8 (=) [3]
- IEEE Spectrum (embedded): 9 (=) Score: 38.8 (=) [3]

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

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

[3] https://spectrum.ieee.org/top-programming-languages/

Source-code Hosting with Ada Build Tools?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Subject: Source-code hosting with Ada build tools?
Date: Fri, 25 Mar 2022 18:56:56 +0200
Newsgroups: comp.lang.ada

I'm planning to move a biggish Ada project from being hosted on my own website to some hosting service, such as GitHub or OSDN. Are there any such services that, in addition to a source-code repository, bug reporting, etc., also offer access to Ada compilers (that is, gnat) for building the SW, ideally on several platforms?

At the moment, my main candidate is OSDN, but they explicitly do not provide any compilers.

TIA for any suggestions, whether with build tools or without.

From: Simon Wright <simon@pushface.org>
Date: Fri, 25 Mar 2022 21:00:58 +0000

Github Actions do this; though I've never set them up for myself.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 26 Mar 2022 21:15:16 +0200

> Github Actions do this; though I've never set them up for myself.

Are you sure that they provide Ada compilers that can be called in an Action?

I tried to find out on the GitHub website, but could not find any list of all the supported languages, and the specific languages they mentioned did not include Ada, and the Search function found nothing about Ada compilation.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 25 Mar 2022 21:01:31 +0000

> Github Actions do this; though I've never set them up for myself.

AdaCore has one for GNAT.

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 26 Mar 2022 21:04:13 +0000

> Are you sure that they provide Ada compilers that can be called in an Action?

https://github.com/marketplace/actions/ada-actions-toolchain

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 26 Mar 2022 23:46:26 +0200

> https://github.com/marketplace/actions/ada-actions-toolchain

Thanks! Looks like GitHub will be my choice, although I am usually a bit Microsoft-allergic.

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 27 Mar 2022 16:59:47 +0100

> Thanks! Looks like GitHub will be my choice, although I am usually a bit Microsoft-allergic.

Aren't we all?

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 29 Mar 2022 21:38:09 +0300

I have my Ahven library and other things at Sourcehut.org: https://hg.sr.ht/~tkoskine/ahven/

They offer generic build service also. For example see one build log from Ahven: https://builds.sr.ht/~tkoskine/job/675294

The build configurations are Yaml files: https://hg.sr.ht/~tkoskine/ahven/browse/.builds?rev=tip

Of course, the software on the build service is limited to open source operating systems and compilers (Linux, *BSDs, GNAT).

Commercial Ada compilers (like ObjectAda or Janus/Ada) are not supported.

For commercial Ada compilers, I run internal homelab network with Jenkins master on RPi4 and couple of Windows build slaves, which fetch the source code from Sourcehut periodically.

And before starting to use Sourcehut, read the caveats page: https://sourcehut.org/alpha-details/

I also think that Sourcehut doesn't support hosting of "random" binaries, like hand-crafted release tar balls. These kinds of things I locate on a separate virtual server.

Ada-related Tools

HAC v.0.0996

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.0996
Date: Sat, 22 Jan 2022 01:41:09 -0800
Newsgroups: comp.lang.ada

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

Web site: http://hacadacompiler.sf.net/

Source repositories:

#1 svn: https://sf.net/p/hacadacompiler/code/HEAD/tree/trunk/

#2 git: https://github.com/zertovitch/hac

* Main improvements since v.0.095:

- range checks on discrete subtype assignment (:=) and conversion
- short-circuit logical operators: "and then", "or else"
- for S = Scalar subtype: S'First, S'Last, S'Succ, S'Pred, S'Pos, S'Val, S'Image, S'Value, S'Range attributes
- for A = array object or array subtype: A'First [(N)], A'Last [(N)], A'Range [(N)], A'Length [(N)] attributes
- "&", "<", ">", "=", "/=" operators defined for the String type (additionally to HAL.VString type)
- CASE choices admit ranges
- forward declarations for subprograms

Enjoy!

PS: for Windows there is an integrated editor that embeds HAC: LEA: http://l-e-a.sf.net

HAC v.0.1

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.1
Date: Sat, 14 May 2022 05:35:55 -0700
Newsgroups: comp.lang.ada

[Omitted common info to previous HAC announcements. —arm]

* Main improvements since v.0.0996:

- packages and subpackages are now supported
- modularity: packages and subprograms can be standalone library units, stored in individual files with GNAT's naming convention, and accessed from other units via the WITH clause
- validity checks were added for a better detection of uninitialized variables.

Package examples and modularity tests have been added. Particularly, a new PDF producer package with a few demos is located in the ./exm/pdf directory.

Enjoy!
Gautier

PS: for Windows, there is an integrated editor that embeds HAC:

LEA: http://l-e-a.sf.net

PPS: HAC will be shown at the Ada-Europe conference (presentation + tutorial)

http://www.ada-europe.org/conference2022/

From: Doctor Who <doc@tardis.org>
Date: Sat, 14 May 2022 18:05:55 +0200

Which subset of the Ada language is covered?

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 14 May 2022 22:24:04 -0700

> which subset of the Ada language is covered?

Quoting from ./doc/hac.txt (section "Language subset"):

"The available Ada language subset supported by HAC is so far, roughly, the "Pascal subset", plus tasking, plus packages, less pointers. From a different perspective, HAC supports Ada 83, less pointers, less generics, less unconstrained types, plus a few items from Ada 95 and 2005. Recursion and nested subprograms are supported."

and: "Tasks are implemented, but not working yet."

From: Leo Brewin <leo.brewin@monash.edu>
Date: Sun, 15 May 2022 10:14:41 +1000

I just tested this on macOS Monterey 12.3.1 and it works perfectly out of the box (as expected for Ada code :).

Great work Gautier!

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Sun, 15 May 2022 02:39:36 +0100

> I just tested this on macOS Monterey 12.3.1 and it works perfectly out of the box (as expected for Ada code :)

You beat me to it by an hour!

> Great work Gautier!

Ditto.

HAC v.0.2

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.2
Date: Sat, 25 Jun 2022 00:43:14 -0700
Newsgroups: comp.lang.ada

[Omitted common info to previous HAC announcements. —arm]

* Main improvements since v.0.1:

- a program run by HAC can exchange data with the program running HAC, through dynamically registered call-backs
  - see package HAC_Sys.Interfacing and demos: src/apps/exchange_native_side.adb src/apps/exchange_hac_side.adb
- the compiler checks that all choices in a CASE statement are covered
- the compiler performs more compile-time range checks and optimizes away useless run-time checks when it's safe to do so.

AdaStudio-2022 Rel. 12/04/2022 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce: AdaStudio-2022 release 12/04/2022 free edition
Date: Wed, 13 Apr 2022 01:19:51 -0700
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2022.

It’s based on Qt-6.3.0-everywher opensource (expanded with modules from Qt-5.15: qtgraphicaleffects qtgamepad qtspeech qtx11extras qtwinextras), VTK-9.1.0, FFMPEG-5.1, OpenCV-4.5.5, SDL2-2.0.20, MDK-SDK (wang-bin) Qt6ada version 6.3.0 open source and qt6base.dll, qt6ext.dll (win64), libqt6base.so, libqt6txt.so (x86-64) built with Microsoft Visual Studio 201 x64 Windows, gcc amd64 in Linux. Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux amd64 Debian 11.1 AdaStudio-2022 includes next modules: qt6ada, vtkada, qt6mdkada, qt6cvada (with face recognition) and voice recognizer.

Qt6Ada is built under GNU LGPLv3 license

https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada modules for Windows, Linux (Unix) are available from Google drive https://drive.google.com/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

WebPage is https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/index.html

[List of detailed files omitted. —arm]

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

The simple manual how to build Qt6Ada application can be read in "How to use Qt6ada.pdf"

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

Leonid (leonid.dulman@gmail.com)

Generic Image Decoder v.10

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: Generic Image Decoder v.10
Date: Sun, 17 Apr 2022 03:26:25 -0700
Newsgroups: comp.lang.ada

There is a new release of GID - the Generic Image Decoder.

Home page: http://gen-img-dec.sf.net/

Project page #1: http://sf.net/projects/gen-img-dec/

Project page #2: https://github.com/zertovitch/gid

New in V.10

* Added a decoder for the QOI (Quite OK Image) format
* Added an "all RGB" demo

About GID

The Generic Image Decoder (GID) is an Ada package for decoding a broad variety of image formats, from any data stream, to any kind of medium, be it an in-memory bitmap, a GUI object, some other stream, arrays of floating-point initial data for scientific calculations, a browser element, a device...

Animations are supported.

Features

* Standalone (no dependency on other libraries, bindings...)
* Unconditionally portable code: OS-, CPU-, compiler- independent code.
* Multi-platform, but native code built
* Task safe
* Endian-neutral
* Use of generics and inlining at multiple nesting levels for fast execution
* Free, open-source

Currently supported formats are: BMP, GIF, JPEG, PNG, PNM (PBM, PGM, PPM), QOI, TGA.

SparForte 2.5

From: Ken Burtch <koburtch@gmail.com>
Subject: ANN: SparForte 2.5
Date: Wed, 27 Apr 2022 06:00:12 -0700
Newsgroups: comp.lang.ada

SparForte is my Ada-based shell, scripting language and template engine.

Version 2.5 is available from www.sparforte.com.

Changes since 2.4:

New features/examples: 23

Changes: 8

Fixes: 26

Known Issues:

On Raspian Bullseye, the calendar package has rounding errors. Possibly due to increased precision of time values in the kernel.

On FreeBSD 13, "environment corrupt" errors are being reported when the spar command runs another spar command. Possibly due to out-of-data GCC Ada for FreeBSD.

Tab completion does not work correctly on directory names containing spaces.

Change Log can be viewed here: https://www.sparforte.com/news/2022/news_apr2022.html

A summary of new features can be viewed here: https://www.pegasoft.ca/coder/coder_january_2022.html

SparForte is my hobby and is built with the support of volunteers. It is open source and is about 123,000 lines of code. It has been in development since 2001.

GCC 12.1.0

From: Simon Wright <simon@pushface.org>
Subject: ANN: GCC 12.1.0
Date: Wed, 11 May 2022 17:58:26 +0100
Newsgroups: comp.lang.ada

Find GCC 12.1.0 & tools for Intel silicon (will run on M1 silicon under Rosetta) at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-x86_64

Built on High Sierra with Python 3.8 (because Apple have withdrawn 2.7 in Monterey).

GCC 12.1.0 for Apple Silicon (aarch64)

From: Simon Wright <simon@pushface.org>
Subject: [ANN] GCC 12.1.0 for Apple silicon (aarch64)
Date: Fri, 27 May 2022 14:05:19 +0100
Newsgroups: comp.lang.ada

Find at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-aarch64-1

SweetAda 0.10

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda 0.10
Date: Thu, 12 May 2022 14:54:29 -0700
Newsgroups: comp.lang.ada

I've just released SweetAda 0.10.

SweetAda is a lightweight development framework to create Ada systems on a wide range of machines. Please refer to https://www.sweetada.org.

Release notes @ https://www.sweetada.org/release_notes.html.

Downloads available @ https://sourceforge.net/projects/sweetada.

This release comes with a huge cleanup of the whole system, with many changes in all areas. The build system seems pretty efficient and stable, with no redundant actions, and is able to accommodate a large set of configuration.

The profile agrees with Ravenscar, and all platforms tested run OK, albeit many of them in a very simple manner. Interrupt handling is for some CPUs still only a placeholder, but many of them are able to handle at least a simple timer in order to have a raw notion of time.

There is a Monitor module (very exemplary) to do user interaction and the Srecord module that could be used as a built-in tool to execute fragments of code. The Time module should provide basic capabilities in order to manipulate a datetime.

Many other changes, large cosmetic refinements and an improved documentation. Syntax changes to adhere Ada 2012/202x and some generics removed from I/O layers to simplify the code and gain speed.

With SweetAda 0.10, I also provide new toolchains based on GCC 11.3.0 (release-20220429), you can find them at SweetAda home or at SourceForge. QEMU emulators are bumped to 7.0.0 (release-20220504).

Unfortunately, I can no longer provide OS X toolchains due to increasing difficulties in building the software (GCC and LLVM disagree on the syntax of some CPU instructions), and lack of time, sorry. But this shouldn't be a problem since SweetAda should be toolchain-agnostic.

Simple Components v4.62

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v4.62
Date: Sat, 21 May 2022 12:03:59 +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:

- Pipe stream implementation added;
- GNAT 12.1 bugs worked around in several package, in particular, in GNAT.Sockets.Server;
- Bug fix in Generic_Set procedure Replace, parameter Updated unset;
- Bug fix in Tables.UTF8_Names procedure Replace, parameter Offset unset under circumstances.

PragmAda Reusable Components

From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Reminder] The PragmAda Reusable Components
Date: Wed, 1 Jun 2022 12:23:50 +0200
Newsgroups: comp.lang.ada

The PragmARCs are a library of (mostly) useful Ada reusable components provided as source code under the GMGPL or BSD 3-Clause license at https://github.com/jrcarter/PragmARC.

This reminder will be posted about every six months so that newcomers become aware of the PragmARCs. I presume that those who want notification when the PragmARCs are updated have used Github's notification mechanism to receive them, so I no longer post update announcements. Anyone who wants to receive notifications without using Github's mechanism should contact me directly.

A New Era for Ada/SPARK Open Source Community

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: AdaCore Blog: A New Era For Ada/SPARK Open Source Community
Date: Mon, 6 Jun 2022 01:56:27 -0700
Newsgroups: comp.lang.ada

I am sharing this here for people who might not have seen it yet:

https://blog.adacore.com/a-new-era-for-ada-spark-open-source-community

[The blog post announces the relicensing of several AdaCore libraries as Apache 2.0 and the discontinuation of the Community Edition in favor of the FSF branch distributed through Alire. —arm]

I will be at the Ada-Europe conference next week if someone wants to talk live about these announcements.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 06 Jun 2022 09:03:25 -0700

Thanks for posting this.

I hope this will reduce the differences among the GNAT versions available in the various OS releases; they've been causing headaches for Emacs ada-mode. But probably not.

Alire 1.2.0

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: [ANN] Alire 1.2.0
Date: Mon, 6 Jun 2022 01:57:09 -0700
Newsgroups: comp.lang.ada

The new release is here:
https://github.com/alire-project/alire/releases/tag/v1.2.0

Adalog Components

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] Adalog components
Date: Mon, 27 Jun 2022 18:23:18 +0200
Newsgroups: comp.lang.ada

I have added a new component to ease processing of CSV files. Moreover, I have renamed package Debug to Tracer, there were too many packages called Debug around!

More details from Adalog's components page:

https://adalog.fr/en/components.html

Ada and Operating Systems

Building GNAT-FSF on FreeBSD

From: William <william@sterna.io>
Subject: Building GNAT-FSF on FreeBSD
Date: Sat, 12 Feb 2022 21:09:54 +0100
Newsgroups: comp.lang.ada

I did succeed to build a modern gcc (with Ada-GNAT FSF) on my FreeBSD 13.0 serveur. :-)

Fernando Oleo Blanco was very inspiring to me (NetBSD porting), and I use Simon J. Wright portings to macOS for my Hackingtosh.

So I decided to do it too!

For now I did a quick try with plain gcc «out of the box»: (story short)

1. Install gcc6-aux pkg from FreeBSD port (2014 -- Last Updated on 2022-01-26). (see also http://www.dragonlace.net)
2. get gcc 10.3 src from GNU.org and compile it with gcc6-aux (gnat compiler seems OK)
3. get gcc 11.2 src from GNU.org and compile it with the just installed gcc/gnat 10.3

In the first place I thought it would not be successful ...

Now it's time to build) and run the ACATS 4.1y

I took a look at Simon’s ACATS Testsuite on SourceForge, but I need to understand those automated scripts.

I’d like to parallelise a maximum of ACATS sub-projects in order to reduce time.

WIP!!

See you later, William

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Feb 2022 09:52:31 +0000

The section "Testing in GCC" in the README tells how to run the tests within the GCC framework that allows parallel running. Note, you'll probably have to hammer C-c to abort a parallel run, the script doesn't respond well to that.

I would have liked to get parallelising working, but those scripts! eww!

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Feb 2022 11:45:05 +0000!

See this thread:
https://gcc.gnu.org/pipermail/gcc/2018-July/226729.html

[This thread discusses how to interrupt Ada tests, as Ctrl-C fails sometimes. –arm]

I'm not sure, but I think that GCC/Ada folk regard the ACATS (2.6, I think) in GCC as more of a confidence thing (DEC used to call it an IVP, Installation Verification Procedure) than a full check.

[...]

Ada Inside

Controlling ST7789 Screen on a RPi Pico

From: Björn Lundin <b.f.lundin@gmail.com>
Subject: Controlling st7789 screen from Ada on a rpi Pico?
Date: Tue, 15 Feb 2022 22:18:44 +0100
Newsgroups: comp.lang.ada

So, I got my first Raspberry Pico :-)

I also got a 'Pico Explorer Base' device at https://shop.pimoroni.com/products/pico-explorer-base

This thing has a st7789 screen. I got it to work with Python.

Now - I see that there is work done with the Pico and Ada - the https://pico-doc.synack.me seems to be a good place to start.

I wonder if there is any port done already for this screen in Ada? Google points me to some python and some c/c++ implementations (whereof Pimoroni's Github has some)

I also came across uGUI http://embeddedlightning.com/ugui/ which looks interesting. Same question there. Ada-port?

I hesitate to start translating one of the c-libraries - butI probably will when time permits if nothing is already in place.

From: jer...@synack.me <jeremy@synack.me>
Date: Tue, 15 Feb 2022 18:03:26 -0800

The Pimoroni Picosystem uses a ST7789 screen, I have a driver for it in picosystem_bsp: https://github.com/JeremyGrosser/picosystem_bsp/tree/master/src

I didn't implement every feature or video mode that the controller supports, so you may need to modify it to suit your needs.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Wed, 16 Feb 2022 08:19:07 +0100

Perfect - just what I was looking for - thanks.

And thanks for the effort of bringing Ada to the Pico

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Fri, 18 Feb 2022 01:31:20 -0800

> I also came across uGUI <http://embeddedlightning.com/ugui/> which looks interesting

I have an Ada binding [1] for the excellent lvgl GUI library [2]. You can get it from Alire: [3].

It is not in a very beginner friendly shape, but it works. I am trying to do a new version that should be easier to integrate into existing projects.

Don't hesitate to say hello on the Ada Gitter chat if you want a little help setting it up.

[1] https://github.com/Fabien-Chouteau/lvgl-ada

[2] https://github.com/lvgl/lvgl

[3] https://alire.ada.dev/crates/lvgl_ada.html

Ada in James Webb Space Telescope? (Cont.)

[Refer to AUJ 43-1: Ada in James Webb Space Telescope? —arm]

From: 姚飞 <yaofei509@gmail.com>
Subject: Re: is Ada used in James Webb Space Telescope software?
Date: Sat, 23 Apr 2022 02:17:05 -0700
Newsgroups: comp.lang.ada

> Interesting. I hadn't heard of the MA31750 but it appears to be a 16 bit processor that implements the MIL-STD-1750A instruction set(!), which I didn't know about either. Apparently it was made in the 1980s but has since been superseded by SPARC architecture cpu's.

MAS31750 + XGC M1750-Ada is a very wonderful combination, we use them for several large satellites, and they are working in orbit now.

Ada and Other Languages

Comparing Languages wrt Energy, Speed, and Memory Use

From: Jerry <list_email@icloud.com>
Subject: Comparing languages wrt energy, speed, and memory use
Date: Sun, 20 Feb 2022 14:59:29 -0800
Newsgroups: comp.lang.ada

This paper comparing 27 languages with respect to energy use, speed, and memory use is interesting. Of course Ada fares very well.

https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf

It is linked from this Slashdot page which I'm sure is full of useless chatter.

https://developers.slashdot.org/story/22/02/20/0143226/is-it-more-energy-efficient-to-program-in-rust

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 22 Feb 2022 21:10:15 +0100

I am going to leave a few comments regarding this paper that I believe everybody should know. Most if not all of these points are known and have been discussed pretty much everywhere; but a lot of people still don't know them or decide to not know.

The programs are taken from the Programming Language Benchmark Game. It is a really cool place that has been providing relevant performance data for a lot of languages and comparisons between them.

Here are a few issues:

1. Quite a few languages are not using heavily optimised code. Ada is one of them. Some of those programs are written as direct translations from other languages from people that did not know the target language.
2. Quite a few of those implementations have not been touched in years. Some of the improvements that may have taken place in the language/compiler/tools may not be taken advantage of. For example, the Ada examples are compiled with -gnatNp. Can anybody say what that flag does? x)
3. C/C++/Rust programs are competing on getting the best results. Other languages are lagging behind. For example, Fortran could do much better. For a couple of years, the Fortran community has been improving the code little by little and they have managed to improve their results.
4. There are a few controversies. Some languages are not allowed to use higher performance libraries while others are allowed their stl or equivalent that do actually use the same tools as those libraries. There are a few other examples.

As the very Game page says, do not take the benchmark seriously. But the communities whose languages are on top, they do not care. Ada has been left behind since very few or nobody is actually taking a look at the code and optimising it...

We may want to improve some of these tests as a community :)

Here are some relevant links:

- Benchmark game: https://benchmarksgame-team.pages.debian.net/benchmarksgame/

- Source code: https://salsa.debian.org/benchmarksgame-team/benchmarksgame

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 22 Feb 2022 21:49:25 +0100

> [good remarks snipped]

Let me add another one: this benchmark does not consider the energy (electrical and human) needed to write and debug the program... That could also make a difference for Ada!

Real ecological balance, taking everything into account, is tricky...

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Thu, 24 Feb 2022 08:42:40 +0100

> Here are a few issues:

One issue is Isaac Gouy's clever approach. (Not complaining. I sometimes didn't see the point, though, of adopting another new thing. For example, when a new regex library was introduced (at some point) that wins hands down by using optimization techniques you'd associate with JIT compilers or with data based optimization. Worth knowing about, but how does it help comparing languages when all you can do is link it to your program?)

> 1. Quite a few languages are not using heavily optimised code

Can you be specific? For example, at least one program currently leads by making extensive use of x86 intrinsic ops.

Some use OMP with intrinsic 128bit ops. Does GNAT have a similar parallel loop in the language yet?

> 2. Quite a few of those implementations have not been touched in years.

Yet, some Ada program versions #N+m used to run faster than #N. They now have their speed difference wiped out or even reversed... I see -march=ivybridge now, so the hardware has likely changed.

> For example, the Ada examples are compiled with -gnatNp. Can anybody say what that flag does? x)

GNAT User's Guide explains. (su-p-press and front end i-N-lining)

> 3. C/C++/Rust program are competing on getting the best results. Other languages are lagging behind. For example, Fortran could do much better.

How would Fortran do much better? Can Ada learn from that?

[...]

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 24 Feb 2022 10:13:46 +0100

> Can you be specific? For example, at least one program currently leads by making extensive use of x86 intrinsic ops.

> Some use OMP with intrinsic 128bit ops. Does GNAT have a similar parallel loop in the language yet?

Yes, take a look at https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gnat-2.html

it is taken from the Pascal implementation and uses intrinsics. My point is that some of these programs are not very Ada-like. As far as I remember, there was one ported from Lua.

Ada 2022 will have a parallel keyword. However, it is still not supported in FSF GNAT, which is the one being used. Also, the benchmarks are Ada 2012.

> GNAT User's Guide explains. (su-p-press and front end i-N-lining)

Correct, but that switch has been deprecated for years, it is no longer documented anywhere in the new GNAT releases:

https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gnat_ugn.pdf

> How would Fortran do much better? Can Ada learn from that?

Fortran is using Intel's compiler, which is known to be one of the best. Fortran compilers can much more easily generate SIMD code and parallelise loops automatically if the code is idiomatic.

Also, Fortran was not fourth in the race a while ago. For example Ada overtook Fortran for a small while. December 2018:

https://web.archive.org/web/20181204085050/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fast.html

Ada is fourth; while it was fifth in April of that same year

https://web.archive.org/web/20180406194535/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html

A year later, December 2019, Fortran could be fourth if it were not for that outlier

https://web.archive.org/web/20191225172425/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html

These are the current results:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/box-plot-summary-charts.html

Take a look at the evolution of the language podium. It has always been C/C++/Rust, but starting from the fourth position there has been quite a bit of rivalry.

[...]

Some Ada programs could use better algorithms, data structures, more up-to-date syntax and parallelism. Some programs could also be made a bit prettier.

The crux of the issue is that you can pretty much always get peak performance for non-GC languages if you use the same techniques, libraries, algos, state of the art compilers, etc. And in a lot of real world cases, even GC languages are not an issue, see Go, Erlang, Julia, Lisp (SBCL), Nim...

But as someone (I believe it was the dean of TUM (Technische Universität München)) once said: "Everybody knows that rankings are flawed, but it is always better to be on top." The benchmark game is, after all, a game. But some people took it too seriously. It is just like Football hooligans.

From: 25.Bx943 <25bz493@nada.net>
Date: Sat, 26 Feb 2022 22:31:19 -0500

After 30+ years, I started messing around with FORTRAN again. One of the things I noticed in the various help notes online was that programmers were actually comparing the numbers of cycles and executables size for various ways of solving any particular problem.

This sort of thinking is rarely seen these days except in the microcontroller universe - and less even there because the RAM/ROM and speed of those devices has increased.

Ada is another language where overall "efficiency" gets at least some consideration.

With energy costs rising, maybe it's time to see MORE of these discussions and comparisons. Global warming be damned - this is a MONEY issue :-)

Oh, and rising power costs may disappear the crypto sector. Those boxes full of GPUs calculating like mad - the power usage is stupendous. Once the energy-in begins to exceed the value of the Bitcoins-out - it's all over.

From: Robin Vowels <robin.vowels@gmail.com>
Date: Sun, 27 Feb 2022 00:05:48 -0800

> This paper comparing 27 languages with respect to energy use, speed, and memory use is interesting.

Has this anything to do with reality?

What of the design, testing, and maintainability of programs?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 27 Feb 2022 09:56:12 +0100

> What of the design, testing, and maintainability of programs?

There are a couple of obvious problems with this study. First, the same data structures, algorithms, and checks for validity of input and so on, in any imperative language, should give very similar machine code. Robert Dewar famously had a collection of equivalent Ada and C programs that produced identical machine code when compiled with gcc. The kind of differences reported between C and Ada or C++ shows that they are comparing apples to orangutans.

Second, there are hard data that show that, compared to low-level languages like C, Ada requires 1/2 the effort to reach deployment, and 1/40 the effort to correct post-deployment errors. The energy consumption for that additional effort should swamp the kind of small differences during execution that this study concentrates on.

Ruby and Ada

From: Mockturtle <framefritti@gmail.com>
Subject: Ruby and Ada
Date: Sat, 14 May 2022 01:46:06 -0700
Newsgroups: comp.lang.ada

As you can guess, my language of choice is Ada, but for small things (often "fast and dirty") or to extract stuff from text files, I use Ruby which I prefer over its direct competitor (much more popular) Python.

Then I read this [1]

> Its [of Ruby] creator, Yukihiro “Matz” Matsumoto, combined parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp)

This could explain the affinity... (Matz is an Adaist! :-))

[1] https://dev.to/rodmatola/ruby-the-best-language-for-general-automation-gh3

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 14 May 2022 05:53:21 -0700

> [...] for small things [...]

BTW: for the small things you describe, you could be tempted by HAC (see the post about HAC a few hours later) :-)...

From: Sromatic <sriviere17@gmail.com>
Date: Thu, 19 May 2022 00:40:03 -0700

I second that.

We wrote in HAC more than 10K lines of code for about 50 scripts (the biggest ones being 3000 lines, the smallest ones less than 20 lines).

I know Ruby (also very nice for small jobs) but HAC is much better (1) and certainly faster too (2)... We also coded a lot in Bash (there are sysadmin here too :)

(1) One of the great things about HAC is that all HAC code can be compiled by GNAT.

(2) HAC is 7 times faster than Bash

And, recently, HAC handles packages... This allows us to have modularity in the Ada way... HAC is a golden nugget ;)

From: Robin Vowels <robin.vowels@gmail.com>
Date: Sat, 14 May 2022 06:40:44 -0700

> [...] for small things [...]

Whether it's small and dirty or something big, PL/I is a great all-rounder.

When Ada Was the Most Popular Language

From: Nasser M. Abbasi <nma@12000.org>
Subject: The good old days, when Ada was the most popular language
Date: Sat, 28 May 2022 06:46:15 -0500
Newsgroups: comp.lang.ada

Check out this cool video

"Most Popular Programming Languages 1965 - 2019"

https://www.youtube.com/watch?v=Og847HVwRSI

At 1:47

1986. Ada was the most popular programming language! (before C took over)

Who Needs Types? Types Make Code Ugly.

From: Nasser M. Abbasi <nma@12000.org>
Subject: who needs types? Types makes code ugly.
Date: Wed, 1 Jun 2022 22:21:08 -0500
Newsgroups: comp.lang.ada

So Ada had it wrong all the time it seems. From

https://python.land/python-tutorial

In a strongly typed language, you need to specify the exact type of each variable, like String, int, and float. It gets even uglier when objects are involved.

Now let’s look at Python variables. In Python, we can do exactly the same without types:

my_name = "Erik"

my_age = 37

my_salary = 1250.70

As you can see, the Python variant is a lot cleaner and easier on the eyes!

And about possible error, they defend this by saying:

In addition, you’ll find out soon enough during testing and fix the error before the software ever goes to production.

So, I think all that Ada needs is to simply remove all those ugly types from the language and it will become popular like Python is now :)

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 2 Jun 2022 12:47:25 +0200

> In addition, you’ll find out soon enough during testing and fix the error before the software ever goes to production.

Proven beyond a shadow of a doubt by the total absence of security vulnerabilities in production S/W.

From: Ldries46 <bertus.dries@planet.nl>
Date: Thu, 2 Jun 2022 13:47:14 +0200

As someone who has been programming since 1966 I used several different languages, Algol 60, Fortran, Basic, C/C++ and Ada, I like using strong types because the ugliest faults you can create are the ones where you by accident use different types in the input or the output of a formula. Such a fault can work through the complete program and result in very tough error searching. Even when the basic failure is using an integer instead of a real.

From: Ben <ben.usenet@bsb.me.uk>
Date: Thu, 02 Jun 2022 16:02:58 +0100

The terms being using in this thread might need to tightened up a bit because I think you are talking about strong /static/ typing.

Python is strongly typed (though exactly how "strong" is debatable) but the checking is at run-time, so you have to rely on testing rather than the compiler. (I don't know enough about Python's new static type syntax to know how strong that is, but it's optional anyway.)

Also, the OP is talking about removing all those messy types, and that's not necessarily the same as removing type checking, either static type checking or at run-time. Haskell, for example, has strong static type checking, but a lot of Haskell is written without ever using a type because of the language's type inference mechanism.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Thu, 02 Jun 2022 10:28:44 -0700

That's just one tutorial. It likely doesn't reflect the views of most Python programmers. The "python.land" site has no official connection

And for what it's worth, Python 3.5 added support for "type hints".

https://docs.python.org/3/library/typing.html

From: John Perry <devotus@yahoo.com>
Date: Thu, 2 Jun 2022 15:10:57 -0700

> (I don't know enough about Python's new static type syntax to know how strong that is, but it's optional anyway.)

I think you mean "type hints"? The compiler doesn't check even when you specify the types. The typing is available for those who want to use a 3rd party tool to do "stuff" with it. See the note at the top of this page: https://docs.python.org/3/library/typing.html

The Python runtime does not enforce function and variable type annotations.

They can be used by third party tools such as type checkers, IDEs, linters, etc.

> but a lot of Haskell is written without ever using a type because of the language's type inference mechanism.

Correct me if I'm wrong, but do you mean "without ever *specifying* a type"? Several recent languages have taken this up, including Kotlin and Rust, though you have to specify some types.

Even Ada 2022 offers it with the "renames" keyword.

From: Ben <ben.usenet@bsb.me.uk>
Date: Fri, 03 Jun 2022 01:02:27 +0100

> Correct me if I'm wrong, but do you mean "without ever *specifying* a type"?

"Use" was not at all the right word since writing 1+2 obviously "uses" types, but I don't mean specify either since types can be specified simply by writing literals like "abc". I should have said something more technical like "without writing any type signatures".

> Several recent languages have taken this up, including Kotlin and Rust, though you have to specify some types.

Yes, and even C++.

> Even Ada 2022 offers it with the "renames" keyword.

I don't know much about Ada newer than about 1990. I'll take a look...

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Thu, 02 Jun 2022 23:37:51 -0400

>I don't know much about Ada newer than about 1990. I'll take a look...

My condolences -- taken literally, that means you are working with Ada-83 (ANSI/Mil-Std 1815A -- later ISO-8652:1987). The first significant update was Ada-95 (and Air Force funded original GNAT).

From: Ben <ben.usenet@bsb.me.uk>
Date: Fri, 03 Jun 2022 19:13:50 +0100

> My condolences

Thanks, but I'm fine. Knowing was not intended to imply forced to use.

> -- taken literally, that means you are working with Ada-83 (ANSI/Mil-Std 1815A -- later ISO-8652:1987).

That was the only Ada I knew, though I knew about the updates of course. Couldn't find any reference to type inference though. Is there a good place to go for a "summary of changes" between standards?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 3 Jun 2022 21:34:36 +0300

> Couldn't find any reference to type inference though.

As far as I know, the only type inferencing that occur in Ada is in for-loops where the type of the loop parameter variable is inferred from the range or container over which the loop iterates.

> Is there a good place to go for a "summary of changes" between standards?

Each version of the Reference Manual has an "Introduction" chapter that contains a subheading "Language Changes", but those are quite terse. If you can find a "Rationale" document for the version in question that usually has much more information about the changes.

For Ada 95:
https://www.adaic.org/resources/add_content/standards/95rat/rat95html/rat95-contents.html

For Ada 2005:
https://www.adaic.org/ada-resources/standards/ada05/

For Ada 2012:
http://www.ada-auth.org/standards/rationale12.html

For Ada 2022, see the Intro in the RM:
http://www.ada-auth.org/standards/ada2x.html

For Ada 2022 I don't think there is any "Rationale" document (yet), but there are various summaries and introductions, for example:
https://learn.adacore.com/courses/whats-new-in-ada-2022/chapters/introduction.html

From: John Perry <devotus@yahoo.com>
Date: Fri, 3 Jun 2022 13:27:38 -0700

> As far as I know, the only type inferencing that occur in Ada is in for-loops where the type of the loop parameter variable is inferred from the range or container over which the loop iterates.

FWIW I was referring to the optional specification of type in a renames clause, which I first read about here:
https://blog.adacore.com/ada-202x-support-in-gnat

(section "Renames with type inference").

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 3 Jun 2022 19:28:23 -0500

> For Ada 2022 I don't think there is any "Rationale" document (yet)

It is not likely that there will be an Ada 2022 Rationale, as no one has stepped up to write it or pay John Barnes to write it. The closest thing we have is the Jeff Cousins overview, which I can't find an on-line reference to (or my copy, for that matter). I'll check with Jeff and hopefully get more information.

From: Wesley Pan <wesley.y.pan@gmail.com>
Date: Fri, 17 Jun 2022 10:33:07 -0700

> no one has stepped up to write it or pay John Barnes to write it.

How much would it likely cost to pay someone to generate the Ada2022 rationale? Maybe the community can join together to help fund the work?

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 17 Jun 2022 13:46:18 -0700

> How much would it likely cost to pay someone to generate the Ada2022 rationale?

Compared to Ada 2012, the 2022 changes look fairly modest.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 17 Jun 2022 21:06:14 -0500

> How much would it likely cost to pay someone to generate the Ada2022 rationale?

Dunno, you'd have to ask John.

I did get a copy of Jeff Cousin's overview that I'll put up on Ada-Auth.org when I get time (probably not until next month).

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Date: Sat, 18 Jun 2022 10:29:02 -0000

>>> It is not likely that there will be an Ada 2022 Rationale

See my follow-up to Randy's June 3 posting quoted above, that I posted on 4 Jun in this newsgroup with subject

"What's new in Ada 2022?" (copied below).

Executive summary:

- John Barnes wrote a 46 page overview on what's new in Ada 2022; it is available as a new appendix in his latest book "Programming in Ada 2012 with a Preview of Ada 2022";
- Jeff Cousin's overview was published in the Ada User Journal (AUJ), and is already available in the online AUJ archive.

Recent addition:

Earlier this week, Tucker Taft presented a very interesting half-day tutorial "Moving up to Ada 2022" at the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022), held in Ghent, Belgium. The event was announced in this newsgroup and via various mailing lists and social platforms. Tutorial participants got a nice overview of what's new in Ada 2022 and practical examples of how to use the new features.

(http://www.adaeurope.org/conference2022/tutorials.html#T1).

To conclude, I repeat below my earlier posting with more information on, and pointers to, John's and Jeff's contributions:

Two additional sources of information on Ada 2022 exist:

- the Ada User Journal;

- the new book by John Barnes.

The Ada User Journal (AUJ, http://www.ada-europe.org/auj/home) has published several articles the last few years about the changes in Ada 2022 (then called Ada 220x).

The latest contribution was the above mentioned overview by Jeff Cousins. It is available from the AUJ online archive:

Ada User Journal, Volume 41, Number 3, September 2020

Jeff Cousins: "An Overview of Ada 202x", pp.159-175

http://www.ada-europe.org/archive/auj/auj-41-3-withcovers.pdf?page=43

And then there's of course the new edition of John Barnes' book: "Programming in Ada 2012 with a Preview of Ada 2022"

https://www.cambridge.org/core/books/programming-in-ada-2012-with-a-preview-of-ada2022/AD30275F35CCECB97EAB80ABC32B019C

Previews of the various sections are available on the cambridge.org site mentioned above, such as the first page of the Preface at https://www.cambridge.org/core/books/abs/programming-in-ada-2012-with-a-preview-of-ada2022/preface/21277D825A1D24906949F642B4AD8BE8

That page includes:

"[...] the main chapters describe the 2016 updated version of Ada 2012 in detail. The book concludes with a major appendix describing the key new features of Ada 2022".

(2016 refers to the year of publication by ISO of the Corrigendum which revised Ada 2012.)

I asked John Barnes about the differences between the original "Programming in Ada 2012" and this new book, apart from the extra appendix on Ada 2022. He provided the following info.

“The main changes are twofold.

In the main body, I have updated it to cover all changes introduced by the 2016 corrigendum. I have corrected all known errors (there were quite a lot) and many cross references were wrong.

An idea of the amount of change can be gathered by noting that the original version had just 6 AIs mentioned in the Index. The new edition mentions 55 AIs in the index.

I also updated the text of the main body to use aspects rather than pragmas where relevant.

So the body is now Ada 2016 although we don't usually talk about that.

The new appendix (46 pages) covers all major features of Ada 2022. The associated website also has things such as the full syntax for Ada 2022 in a style matching the book (that's another 30 pages). Also an updated table of the facilities in containers (14 pages). And some worked examples using new features especially using the big integer packages (currently another 14 pages).

Each chapter of the main book ends with a checklist outlining the new features and referring to the appropriate place in appendix 4 where they are discussed.

-- John Barnes, 14 May 2022, with permission"

I hope this helps.

Dirk Craeynest

From: Paul Rubin <no.email@nospam.invalid>
Date: Sat, 18 Jun 2022 15:16:46 -0700

> To conclude, I repeat below my earlier posting with more information on, and pointers to, John's and Jeff's contributions: ...

> I hope this helps.

Yes, thanks, those references are useful for understanding the changes introduced in Ada 2022. I had thought the idea of a formal rationale was different: not just to explain the changes, but also to explain from an authoritative standpoint why the decisions were made. I don't know how important rationales traditionally have been in the Ada world. But, Ada 2012 introduced a much larger set of changes than Ada 2022 did. So I can understand if a rationale was more important in 2012 than in 2022.

I guess if the higher-end Ada community thought that a 2022 rationale was necessary, they would have required it and funded it. As a not-so-serious user or wannabe user, I don't think I need it, but that's just me.

I do notice long after reading "Ada Distilled" that most of the discussions on this group about technical aspects of Ada still baffle me. So I think a more advanced online tutorial would do some good. I believe the current Ada Wikibook is nice for beginners but doesn't cover more advanced topics all that well.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Jun 2022 16:40:06 -0500

> - Jeff Cousin's overview was published in the Ada User Journal (AUJ), and is already available in the online AUJ archive.

Correct. The version of Jeff's overview I have is quite a bit newer than the version from the AUJ, and has many errors corrected. So I would suggest reading that version rather than the original AUJ version (but of course, only once I can get it posted).

Ada Practice

GtkAda for GTK4?

From: Andreas Almroth <andreas@almroth.com>
Subject: GtkAda for GTk4?
Date: Sun, 13 Feb 2022 07:32:23 -0800
Newsgroups: comp.lang.ada

Looking at the excellent support for GTK in GtkAda over the past many years, which I have enjoyed using, I was looking for (aka googling) references to any initial thoughts/work on having GtkAda to also support GTK4.

I know, GTK4 has only been "out" for a little over a year, but it would be interesting to know if anyone is considering doing this. I would be glad to participate, although with limited know-how of the inner workings of GtkAda, but at least testing perhaps.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 13 Feb 2022 17:46:04 +0100

Well, knowing GTK's disastrous history it cannot be "also", it must be either 3 or 4.

GTK 4 breaks basically everything one could ever think of.

To me new features in GTK 4 do not look worth changing the API again, not even useful, just fancy stuff. It seems that GTK team keep on breaking the API rather out of fun than necessity. Instead of hardening the code. GTK 3 is still buggy as hell.

Of course, at some point one will have to migrate, but how about sitting GTK 4 over and going straight to GTK 5? Unless they lose remaining users...

GtkAda is maintained by AdaCore, so it is them [whom you have] to ask.

From: Andreas Almroth <andreas@almroth.com>
Date: Sun, 13 Feb 2022 12:26:11 -0800

[…]

> Of course, at some point one will have to migrate, but how about sitting GTK 4 over and going straight to GTK 5? Unless they lose remaining users...

Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++ (which I have found cumbersome). Most other GUI frameworks are based on C++, for instance QT. QTAda is as far as I know not maintained (I haven't seen much in a very long time).

[...]

From: Andreas Almroth <andreas@almroth.com>
Date: Mon, 14 Feb 2022 00:45:07 -0800

First, I have to correct myself... Seems Leonid Dulman provides QT5 and QT6 support as part of Ada Studio (google qt6ada). Just saw another post on the adagorge.org re-design, and found QT that way…

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 13 Feb 2022 21:45:57 +0100

> Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++ (which I have found cumbersome).

Absolutely. C API is a huge advantage. However GTK and stuff is monstrous, practically impossible to handle manually.

GtkAda bindings are generated by a tool designed by AdaCore. This tool might require massive changes when migrating to GTK 4.

I cannot speak for AdaCore, but I think any help will be welcome.

Good luck.

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 13 Feb 2022 22:35:44 +0000

>> Well, knowing GTK's disastrous history it cannot be "also", it must be either 3 or 4. GTK 4 breaks basically everything one could ever think of.

It's almost like wxAda would've been better...

> Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++

Can confirm, binding C++ is too easy to burn out on, having done so on wxAda.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 13 Feb 2022 23:46:01 -0800

> GtkAda bindings are generated by a tool designed by AdaCore. This tool might require massive changes when migrating to GTK 4.

I wrote that python script years ago, when the XML files that describe the gtk+ API were actually pretty bad type-wise. The script is full of special cases, and very ugly. I don't think anyone should use it as a basis for binding to gtk 4, it would likely be much better to restart from scratch. I believe the XML files have improved significantly since then, and are used by more language bindings, too, so that could likely be simplified.

From: Andreas Almroth <andreas@almroth.com>
Date: Mon, 14 Feb 2022 00:47:53 -0800

> I wrote that python script years ago [...]

Thanks Emmanuel for your input. Seems it indeed would be a larger effort, and as Dimitry states, perhaps one should wait for the next major release. It will take some time in any event to create the interface binding.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 14 Feb 2022 20:50:20 +0100

> I wrote that python script years ago

This may be worth mentioning...

Fortran also has a GTK binding [1]. It is also autogenerated with a Python script. As far as I can remember, the change from GTK3 to 4 was not too big. It did obviously require changes and a bit of elbow grease, but they had GTK4 support as soon as it became official. The actual code is present here [2].

This may serve as a comparison or reference for what may be needed. It is obvious that the Fortran people did have a different starting point and Fortran is a different language. I am just including it for reference.

[1] https://github.com/vmagnin/gtk-fortran

[2] https://github.com/vmagnin/gtk-fortran/tree/gtk4/src

What Is the Name of the "|" Symbol?

From: Matt Jaffe <matt.jaffe@gmail.com>
Subject: What is the name of the | symbol?
Date: Fri, 25 Mar 2022 12:04:57 -0700
Newsgroups: comp.lang.ada

In using it in a named association array aggregate, its semantic are "and" --- e.g., some_1D_array := (1 | 3 | 7 => 5, others => 10) sets elements 1 and 3 and 7 to the value 5. In a case statement, its semantics are "or" --- e.g. when 1 | 3 | 7 => ... any of the values 1, 3, or 7 for the case expression will select the ... code for execution. Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 25 Mar 2022 23:21:37 +0100

ARM 2.1(15/3)

(http://www.ada-auth.org/standards/aarm12_w_tc1/html/AA-2-1.html#I1201) says its name in Ada is "vertical line".

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 25 Mar 2022 21:24:32 -0700

The Unicode name is U+007C VERTICAL LINE, alias name vertical bar.

/ is U+002F SOLIDUS, alias names slash and virgule. I haven't heard of the name "solidus" used for symbols other than /.

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Fri, 25 Mar 2022 12:16:04 -0700

In using it in a named association array aggregate, its semantics are "and" --- e.g., some_1D_array := (1 | 3 | 7 => 5, others => 10) sets elements 1 and 3 and 7 to the value 5. In a case statement, its semantics are "or" --- e.g. when 1 | 3 | 7 => ... any of the values 1or 3, or 7 for the case expression will select the ... code for execution. Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Fri, 25 Mar 2022 19:23:19 +0000

How about reading it like this (read with a fixed-width font):

a := (1 | 3 | 7 => 5, others => 10)

if the index is one or three or seven then five else ten fi

Similar syntax appeared in Algol 68. | is frequently used for "alternatives" -- it's just a question of what's being referred to. Here, it's all the alternative indexes that map to a specific value.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 25 Mar 2022 22:03:08 +0200

That is a quirk of natural language, where "and" and "or" are used in non-mathematical ways. You could as well describe the aggregate as saying "if the index is 1 or 3 or 7, the element is 5", and you could describe the case statement as saying "this when-branch is executed when the case selector is 1 and 3 and 7".

As '|' is used in some logical formalisms for disjunction ("or"), and in syntactical notation (BNF) to separate alternatives, I tend to read it as "or".

> Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

For the name, see https://en.wikipedia.org/wiki/Vertical_bar , where indeed "vertical bar" seems favoured. However, I'm pretty sure that I have seen "solidus" used, too, but Wikipedia says that is a synonym for "slash" (/). Wiktionary does not recognize "solidus" as a term for any punctuation mark.

From: Chris Townley <news@cct-net.co.uk>
Date: Fri, 25 Mar 2022 23:24:45 +0000

Probably wrong, but for a Unix user since the last century, I call it 'pipe'

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Sun, 27 Mar 2022 11:57:48 -0700

> Probably wrong, but for a Unix user since the last century, I call it 'pipe'

Well, non-judgmental type that I am, I'm not going to say you're "wrong", but pipe is the name and usage for that symbol when programming a Unix shell. Its semantics in Ada are quite different, so I don't think calling it pipe quite fits. (So I guess I'm not a pipe-fitter either ;-)

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 26 Mar 2022 00:58:53 +0000

> Probably wrong, but for a Unix user since the last century, I call it 'pipe'

Or bar.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 26 Mar 2022 17:38:59 -0700

> Or bar.

Emacs ada-mode grammar calls it BAR.

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Sun, 27 Mar 2022 12:01:05 -0700

> Emacs ada-mode grammar calls it BAR.

"Bar" sounds like the best alternative so far; I think that's what I'll use when talking to my students.

Thanks.

From: Chris Townley <news@cct-net.co.uk>
Date: Sat, 26 Mar 2022 02:01:38 +0000

> Or bar.

No that is for after work ;)

Aggregate with (parens) Considered Obsolescent

From: Simon Wright <simon@pushface.org>
Subject: Aggregate with (parens) considered obsolescent
Date: Mon, 11 Apr 2022 17:15:08 +0100
Newsgroups: comp.lang.ada

GCC 12. with the -gnat2022 switch, supports (a large part of) ARM 2022. One of the changes is AI12-0212[1], the use of square brackets [] in array aggregates.

I was surprised to find that the compiler reports the use of parentheses () for array aggregates as obsolescent! To quote PR104751[2],

==========

Compiling

```
procedure New_Syntax is
   T : array (1 .. 5) of Integer;
begin
   T := (1, 2, 3, 4, 5);
end New_Syntax;
```

with -gnat2022 -gnatwj gives

new_syntax.adb:4:09: warning: array aggregate using () is an obsolescent syntax, use [] instead [-gnatwj]

but use of parens is not in Annex J; use of brackets is an option, AARM 202x Draft 32, 4.3.3(49.m).

Having -gnatwj as part of -gnatwa makes this very intrusive.

==========

The fact that it happens with -gnatwa, which is a switch that I suspect quite a lot of us use, will be particularly annoying for those who use -gnatwe (treat warnings as errors) and who want to support multiple compiler releases (for example, the Ada Drivers Library).

The response dismissing the PR suggested using

```
pragma Warnings (Off, "*array aggregate*");
```

and one glimmer of hope is that this can be used as a configuration pragma.

I could remove the problem from macOS releases that I support (sem_aggr.adb:1803..1815), but of course that would lead users into problems when using another GCC 12+ release.

[1] http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0212-1.txt?rev=1.29&raw=N

[2] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=104751

Use Clauses and Naming Schemes

[Offshoot from “Unchecked_Deallocation Usefulness”, in AUJ 42-2, 2021. The conversation went into naming preferences in relation to “use” clauses. —arm]

From: Thomas <fantome.forums.tdecontes@free.fr.invalid>
Subject: use clauses
Date: Wed, 13 Apr 2022 01:25:31 +0200
Newsgroups: comp.lang.ada

> For me, a naming scheme that discourages the use of (package) use clauses is a bonus. (Such a scheme makes it easier to avoid use clauses.)

I agree to avoid use clauses.

(I personally prefer Lists.List, like Vincent Marciante - I like Ada.Containers.* naming :-))

> I personally only use "use type" in new code (there's tons of old code for which that doesn't work, of course, but that doesn't change the principle).

what do you think about:

- "use all type" clauses?

- List.Clear? (could you remember me how you call that, please?)

- List.Clear does work only if List is tagged?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 12 Apr 2022 20:05:00 -0500

> what do you think about:

> - "use all type" clauses?

This is OK; I don't use them mainly because I only use features implemented in Janus/Ada, and "use all type" is not yet implemented there.

The fundamental problem with "use" is that it makes everything visible, and then deals with conflicts by making those things invisible again. That's not a problem for overloadable primitive operations, since the profile is included and conflicts only occur when someone has made a lousy design choice (creating a routine with the same name and profile as a primitive) [Most such conflicts come from maintenance when some existing routine is moved to be primitive; in such cases, the original routine simply should be removed.] Since "use all type" only works on overloadable primitives (and things that work rather like primitives), it's fairly safe. One could make an argument that primitive operations should always be visible when the type is (that's not the Ada rule, but arguably it would work better in most circumstances) - and you should always know to look at primitives anyway when trying to find something..

> - List.Clear? (could you remember me how you call that, please?)

For tagged types, you can use prefix notation, so "My_List.Clear" is the easiest. With "use all type List", you can write Clear(My_List). If your objects have well-chosen names, it's not really needed to have the type around for such operations, even when use clauses are in place. Thus, "Clear", not "Clear_List", and that works well even when someone uses everything in sight (of course, they may have a hard time finding where Clear is defined when debugging, but that's their choice).

> - List.Clear does work only if List is tagged?

Right. There are a number of semantic issues for untagged types, the main ones having to do with implicit dereference (which occurs in this notation, as in any other selected_component notation). If you have a prefix of an access type, it gets very messy to determine which dereference is which. And just allowing composite types doesn't work well either: a private type that is completed with an access type would *lose* operations when it had full visibility -- that seems pretty weird.

It originally got limited to tagged types as that was easy to do and didn't have semantic issues. We were going to look at generalizing the prefix notation again (several people asked about it), but no one made a concrete proposal and it never went anywhere for Ada 2022.

Max Line Length Preferences

From: Thomas <fantome.forums.tdecontes@free.fr.invalid>
Subject: max line length
Date: Mon, 18 Apr 2022 23:58:56 +0200
Newsgroups: comp.lang.ada

How do you set your max line length?

Using indentations a lot, I find that 80 is short. but I don't realize how many people I'm going to disturb if I set a greater length, because I don't know all your uses.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 19 Apr 2022 09:38:00 +0300

I limit lines to 80 characters, because I very often want to use a side-by-side diff of file versions, which means having a window wider than two line-lengths. Text in a 170-character-wide window is still readable, but wider ones are not, for me as an older guy with stiff eye-lenses.

To make do with 80-character lines, I often use local or partial use-clauses, and I divide long calls across many lines, usually having only one parameter per line. By a "partial use clause" I mean, for example, "use Interfaces", when I really need to use Interfaces.C, so I still have to qualify with "C.zzz" but not with "Interfaces.C.zzz".

I also group subsystems into package families (parent and child packages) which means that the children can directly use parent-declared identifiers without qualification.

Other means to keep lines short include using a small indentation step (I now use 3 spaces, but I'm considering changing to 2 spaces) and keeping subprograms short, which also helps the readability.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Apr 2022 16:34:04 -0700

> I limit lines to 80 characters, because I very often want to use a side-by-side diff of file versions,

I prefer top/bottom diff, partly for this reason.

But my monitor can easily display 240 characters across. And I have good glasses.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Tue, 19 Apr 2022 21:30:38 +0200

> how do you set your max line length?

I use the Preferences menu selection in my editor. But that's probably not what you intended to ask. I set mine to 132 characters.

> using indentations a lot, i find that 80 is short.

When I started out, source lines were limited to 80 columns because that was the length of punched cards, but the line printers could print 132 columns. In the 1980s printing switched from 14 x 11 inch paper in line printers to 8.5 x 11 inch paper, but it was still possible to print 132 characters in landscape mode, so that's what I used if I had an editor that could handle long lines easily (screens were not large enough or high enough resolution to be suitable for reading programs, so I still tended to print them when that was needed. Today printing is not needed much, but I continue to use 132 columns. If others want a different line length they may reformat it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 19 Apr 2022 22:07:20 +0200

> how do you set your max line length?

72. I used to program in FORTRAN on punched cards. (:-))

These days I use 3 split GPS Windows side by side.

Then I am using the "use" clause, so I do not need a thousand of characters to just write Z := X + Y; (:-))

> using indentations a lot, I find that 80 is short.

Refactor the code and use local subprograms.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Apr 2022 16:31:56 -0700

> how do you set your max line length?

120 chars; I assume readers have a big display like mine.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Apr 2022 02:57:08 -0500

> I prefer top/bottom diff, partly for this reason.

The diff program I use can scroll sideways if necessary, and so can every editor I've used since 1985, so this isn't generally an important concern. The Janus/Ada source used a "soft" limit of 80, mainly because that's what terminals and PCs displayed back then, but we never broke lines just for that reason. Typically, the indent is more than the overrun anyway (so that actual text never exceeded 80 characters). Of course, one has to break really long calls, like the call to create a window in Claw (which usually has a dozen or so parameters).

Aspect Location in Expression Function

From: Blady <p.p11@orange.fr>
Subject: Aspect location in expression function.
Date: Sat, 14 May 2022 13:47:28 +0200
Newsgroups: comp.lang.ada

I'm puzzled when I want to change a function body with aspects to an expression function, for instance:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  with Pre => S.Valid
  is
  begin
  return S.Length;
  end;
```

have to be changed in:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  is (S.Length)
  with Pre => S.Valid;
```

The location of the aspect has moved to the end.

I'd like simply replace the begin block by the expression, as:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  with Pre => S.Valid
  is (S.Length);
```

What could be any reasons not to permit it?

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 14 May 2022 17:40:03 +0200

What you say is logical if you think of an expression function as a body; however, it is more like a specification (it can appear in a package spec, although it can complete a specification), so the place where the aspect appears makes sense. And it would be confusing to allow the aspect in two different places. It is the same for separate bodies of subprograms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 23 May 2022 23:05:12 -0500

> What you say is logical if you think of an expression function as a body; however, it is more like a specification

To make a functioning :LR grammar for Ada, I *had* to allow the aspect specification in both places, and then make one of them illegal. Which is more work than just allowing in either place. So I guess it is a matter of perspective. :-)

To the OP: we discussed placement of aspect specifications ad-nauseam, as issues like this always were coming up. There is no consistent rule that really works well, because one does not want small things following large sets of aspect specs -- they can get lost and overlooked.

For instance, one puts aspect specifications after "is abstract" as otherwise that could be lost after a lengthy precondition expression (and it's too important to be lost). See how that could happen in the following (illegal) declaration:

```
procedure P (A, B ,,,)
   with Pre => <very long expression that extends over several lines here>
   is abstract;
```

So something like this (and "is null" as well) require the Pre at the end:

```
procedure P (A, B ,,,)
   is abstract
   with Pre => <very long expression that extends over several lines here>;
```

Expression functions generally follow the same rules as the older null procedures, thus they ended up with the same positioning. It's not as obvious a case here, since the return expression can also be long, but we thought it should be consistent.

BTW, I don't think there ever is a reason to go from [a function –arm] with a normal body to an expression function (assuming the body is legal). A normal body is more readable and less of a hassle during maintenance. The advantage of an expression function is to use it in places where a regular body is not allowed and/or just to be lazy writing the body - neither of which would ever require changing *to* an expression function. Maintenance might require changing *from* an expression function if the body has gotten too complex (for instance, needs a variable declaration), but that generally will require moving the function as well so "ease" of doing so isn't very relevant.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Tue, 24 May 2022 20:24:14 +0200

> For instance, one puts aspect specifications after "is abstract" as otherwise that could be lost after a lengthy precondition expression (and it's too important to be lost).

Isn't this emphasis on "is abstract" losing the very point of abstraction?

> See how that could happen in the following

> (illegal) declaration:

> procedure P (A, B ,,,)

> with Pre => <very long expression that extends over several lines here>

> is abstract;

Who cares to see "is abstract" if P is in a spec? The implementer, I guess, but the client? Less so.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 25 May 2022 00:20:50 -0500

> Who cares to see "is abstract" if P is in a spec? The implementer, I guess, but the client? Less so.

Any client that needs to declare an extension (pretty common in OOP), especially as "abstract" routines mostly are used with root types (and interfaces). I suppose you could "program by error" and just let the compiler complain if you don't give a body for something abstract, but it's generally recommended to know what you're doing and not just try to make the compiler happy.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Wed, 25 May 2022 20:45:58 +0200

> Any client that needs to declare an extension (pretty common in OOP),

Another, dare I say, more frequent way of being a client of a type is being a caller of the type's subprograms, such as P, rather than being an implementer of a type's concrete behavior. (The two can overlap, but I'm thinking of the more frequent human clients here :)

A case I'd single out is a type that comes with a factory F. I'd expect the associated type T to be abstract. This goes without saying! ;-) A client needs to know the "behavioral" interface of T and also that of F. The "is abstract" then remains as helpful language technology, but as seen inside the factory.

(So, I'd put "is abstract" last.)

> especially as "abstract" routines mostly are used with root types (and interfaces). I suppose you could "program by error"

Not design errors, but mechanical errors duly output by the compiler. The programmer will be programming by "following the language's rules". IDEs and compilers will assist the programmer who is implementing an abstract type. For example, the usual IDE has this suggestion following its compiler's error message:

 Fix: "Add unimplemented methods"

(for)

 Error: "The type must implement[!] the inherited abstract method ..."

The IDE will do so if you answer "Yes" and programmers can provide their own adjustments to template text that this mechanism will be using. Thus, again, programmers can involve useful language technology in a template's text. I remember some Ada tools offering similar features.

What Is X'Address of an Empty Array?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: What is X'Address of an empty array?
Date: Tue, 31 May 2022 14:19:24 +0200
Newsgroups: comp.lang.ada

I have a language lawyering question. According to ARM X'Address is the address of the first array element. What is the address of an empty array?

In the case of an array with bounds it could be the address following the bounds.

But what about a definite empty array? Of zero length (and presumably zero size). Would the compiler have to invent some address?

P.S. With GNAT:

```
type NUL is array (1..0) of Integer;
S : NUL;
```

S'Size is 8 and it has some address that holds the byte.

Talking about the dark matter in our Universe. This is what empty arrays are constructed of! (:-))

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 31 May 2022 16:35:49 -0500

Aliased objects should never have zero-size in Ada (or, at least, ever be allocated at the same address). I believe that is

because of the required result of the equality operator. Specifically:

```
type NUL is array (1..0) of Integer;
A, B : aliased NUL;
type PNul is access Nul;
PA : PNul := A'Access;
PB : PNul := B'Access;

if PA = PB then
    Report.Failed ("Access to two distinct
     objects cannot be equal");
end if;
```

If an object is not aliased, it is undefined whether 'Address will work reliably with it (it probably does in GNAT, it might not in Janus/Ada, etc.) If the objects ARE aliased, then 'Address works essentially the same as 'Access.

I personally find this a bit of overspecification in Ada, but since zero-size objects are unusual, no one has thought it worth going through the effort to change. (And of course such a change would complicate static analysis.) We (the ARG) did discuss this topic at one point (I don't have the AI number at hand).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 1 Jun 2022 14:49:25 +0200

> I personally find this a bit of overspecification in Ada

It is actually rather nice.

I recently stumbled upon code:

```
A (A'First)'Address
```

in C bindings, when the other side has something like:

```
const double * a, int len
```

It of course fails in the marginal case when the array is empty. But A'Address seems to never do.

Modern Syntax for Complex Conditions

From: Matt Borchers <mattborchers@gmail.com>
Subject: Ada needs some modernization
Date: Tue, 31 May 2022 10:54:46 -0700
Newsgroups: comp.lang.ada

Throughout my career, I often find myself writing code similar to:

if (A and B) or else (not A and C) then...

and I always wished there was a better and clearer way to write this in Ada. Then along came if expressions. But, if expressions don't help that much with readability although it is arguably simpler:

if (if A then B else C) then...

What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement? I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better. These same languages often also have a null check operator A ?? B (where A and B are access types of the same Type) such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Again, I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search. This almost always requires some externally defined variable, like:

```
--assuming arr'First is not Integer'First
found := arr'First - 1;
for i in arr'Range loop
   if arr(i) = match then
      found := i;
      exit;
   end if;
end loop;
if found in arr'Range then
   --do something A
else
   --do something else B
end if;
```

Of course I could move the "do something A" into the if block within the loop, but I still need to know if I must run the alternate code afterward. It would be nice to avoid having to create a variable just to indicate the success state or indexing location found. Maybe something like:

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit;
   end if;
then
   --do something else B
end loop;
```

The "then" part only executes after the loop terminates normally, i.e. only when the loop does NOT exit early by "exit" or "return" statement.

I think syntax enhancements like these could go a long way to making Ada feel like it is at least keeping up with modern languages and I think current programmers expect "ease-of-use" syntax from today's languages. Other contemporary modernized languages have taken ideas from Ada, but Ada has not continued to pioneer ideas as quickly. Perhaps that's by choice or design.

Thoughts?

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Tue, 31 May 2022 12:05:48 -0700

In your proposal, the "do something else B" appears before "end loop", which is not a very intuitive way to indicate a statement happening *after* the loop.

I suspect there is room for improvement...

Perhaps you would like to show an equivalent piece of code in a what you call a "modern language"?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 31 May 2022 21:55:05 +0200

> if (A and B) or else (not A and C) then...

> if (if A then B else C) then...

Not the same. In the former A may be computed twice.

> What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement?

I newer felt it necessary. To me much more aggravating is code that combines test/allocator with renaming, i.e.

```
if P /= null then
   declare
      X : T renames P.all;
   begin
      ...
   end;
end if;

------------

if X in T'Class then
   declare
      XX : T'Class renames T'Class (X);
   begin
      ...
   end;
end if;

-----------

   P : access T'Class := new S;
   X : S renames S (P.all);
```

If one could come up with some syntax for if-then-declare and new-then-declare that would cover a lot of cases.

> I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better. These same languages often also have a null check operator A ?? B (where A and B are access types of the same Type) such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Not in a strongly typed language IMO.

[...]

> Maybe something like:

> for i in arr'Range loop

> if arr(i) = match then

> --do something A

> exit;

> end if;

> then

> --do something else B

> end loop;

I usually use a nested function, e.g. search with a fallback:

```
function Get_Me_Something return
Element is
begin
  for I in arr'Range loop
    if Arr (I) = match then
      return Arr (I);
    end if;
  end loop;
  return Default;
end Get_Me_Something
```

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 31 May 2022 17:46:04 -0500

>What amendment can we suggest to the Ada syntax [...]? I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better.

Which is the rub. Ada is *not* about clever operators that hardly anyone knows what they do. Indeed, the original proposal for Ada 2012 had an "implies" operator. But we quickly found out that there are many people that don't know off-hand what function an implies operator does. We were pretty sure that every Ada programmer would understand an if expression.

Note that pretty much the only place that you should almost never use an if expression is in the choice of an if statement. If you already can write an if statement, you don't need an if expression! If expressions exist to make initializations and assertions like (Pre/Post) easier to write.

So I would never write your expression in the first place (either of them). I'd write something like:

```
if A then
   if B then.
   else
  end if;
else
   if C then
   else
  end if;
end if;
```

The contents of the arms should be short anyway, and typically will just be a procedure call (and possibly some debugging, which is way easier if the conditions are kept simple).

>null check operator A ?? B [...] such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Again, "utility" is not the criteria for Ada, rather understandability for future maintainers is the primary criteria. The last thing we need is a bunch of fancy but little used operators that [leave] someone cold when reading some unfamiliar code. (Yes, of course you can look them up on-line, but stopping to doing so necessarily breaks your train of thought.)

And this construct fits nicely into an if expression, with no magic:

```
(if A /= null then A else B)
```

and this extends nicely to more likely cases:

```
(if A /= null then A elsif B /= null then B
 else raise Program_Error)
```

Personally, I don't believe I've ever written something where such an operator would be useful; one needs to check everything for null (you can't usually can't assume B is nonnull, either). And the fall backs are generally more complex than using some other object. Moreover, probably A should have been declared null-excluding so it doesn't need to be tested in the first place. :-)

> I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search. ...

>for i in arr'Range loop

> if arr(i) = match then

> --do something A

> exit;

> end if;

>then

> --do something else B

>end loop;

>The "then" part only executes after the loop terminates normally, ...

In Ada terms, an exit *is* normal completion, so you would need some different terminology.

> i.e. only when the loop does NOT exit early by "exit" or "return" statement.

We've discussed the "continue" statement multiple times, and have always ended up deciding that we are better off without it. (We've also discussed allowing "exit" from blocks, but that turns into a mess when blocks and loops get mixed, at least if one wants the code to do the same thing in Ada 2012 and in future Ada.)

We've essentially decided that it is better to use a goto in such rare cases. The case you show above is similar.

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      goto Loop_Finished;
   end if;
end loop;
-- We get here if the search item is
--  not found:
--do something else B
<<Loop_Finished>> null;
```

Remember that every feature added to a language adds costs in implementation, documentation, and in tools (analysis, checkers, etc.). A feature needs to be quite useful in order to make the cut.

Aside: in the case above, I've usually written such loops like:

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit;
   elsif i = arr'Last then
     --do something else B
     exit; -- Not really needed, but clearer
           -- what is going on.
   end if;
end loop;
```

I've never been that happy with the duplication of the termination condition, but this avoids any extra objects or any gotos.

If I was going to try to fix your problem with a language feature, I'd probably try to define an attribute to avoid needing to duplicate the termination condition. Something like:

```
Loop_Name: for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit Loop_Name;
   elsif i = Loop_Name'Range'Last then
     --do something else B
     exit Loop_Name; -- Not really needed,
     --but clearer what is going on.

   end if;

end loop Loop_Name;
```

(We probably would allow 'First and 'Last in such a case.) But this technique doesn't really work with user-defined iterators (which don't necessarily have a defined end), and I'm unsure if it is important enough for another whistle.

>"ease-of-use" syntax from today's languages.

Ada has *never* been about "ease-of-use". It is about readability, maintainability, and understandability. (See the “Design Goals” in the Introduction -- http://www.ada-auth.org/standards/2xrm/html/RM-0-2.html.)

Enhancing readability might also enhance ease of use (for instance, user-defined literals, target name symbols, and user-defined indexing all were added to enhance readability by avoiding duplicative text that provides little information), but it is never a primary goal for an Ada feature.

>Other contemporary modernized languages have taken ideas from Ada, but Ada has not continued to pioneer ideas as quickly.

This is not true. Ada pioneers ideas all the time (see delta aggregates, aggregate iterators, the target symbol, parallel stuff, etc. from Ada 2022). What Ada does not do is waver from its core goal of readability and maintainability. So we don't waste time with tiny features that are more likely to harm readability and understandability than help. (Admittedly, what features are really necessary and which are just nice to have is always a personal choice.) Additionally, Ada has always been designed with a "building-block" approach, so we don't provide (say) a semaphore, but rather the tools (the protected type) to write one (and many other constructs). An if expression is a building block; funny boolean operators with limited uses are not.

I personally am not the least bit interested in worrying about ease-of-use gadgets in other languages. If programmers need such gadgets to be comfortable, they probably don't have the right mindset to be great Ada software engineers in the first place. Saving a few characters in a few expressions simply does not matter when compared to the effort needed to define and document a good data abstraction (for instance, an abstract data type and package).

There *are* features that probably would not interfere with Ada goals of readability. One of them that comes up periodically is an "at end" clause so one could write final wishes for a block/subprogram/package without writing a bunch of exception handlers (which doesn't work in the case of abort!) or one-time use controlled types. I'm sure there are others.

And certainly other languages have interesting features that Ada should steal, the Rust owned access types would be an obvious example. (Don't get me started on why Ada 2022 does not have those.) But "ease-of-use" is not interesting, at least when it does not make readability better. (I want people to replace "and" and "or" with if expressions as much as possible, as those are much more understandable. No more operators please!)

Randy.

P.S. Man, did I spend a lot more time than I planned answering this. I hope it helps.

From: John Mccabe <john@mccabe.org.uk>
Date: Wed, 1 Jun 2022 00:24:24 -0700

> P.S. Man, did I spend a lot more time than I planned answering this. I hope it helps.

FWIW, I thought it was valuable. As I read through it I was constantly thinking of how I wish the people tweaking C++ (which, for various reasons, I'm using now) would take the same attitude, rather than trying to feed their own egos by adding all sorts of random rubbish that, due to the current 3 year cycle, tends also to be either temporary or half-baked random rubbish!

Thank you, Randy!

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Wed, 1 Jun 2022 21:00:31 +0200

> What amendment can we suggest to the Ada syntax [...]

What you call "modernization" looks to me a lot like "repeating mistakes that Ritchie made over 50 years ago".

"A ? B : C"? Or is it "A : B ? C"? If only there were a less cryptic, easier to remember and understand way to express it. Something like "(if A then B else C)", for example.

"A ?? B" might be "useful and helpful" if you use (or think in) a language with pointers to objects everywhere, but in a language where such pointers are never needed, like Ada, it is neither, especially since a conditional expression would handle it just fine if it were ever needed.

> I often find myself writing a loop to search ...

When you write something for a second time, it's a signal to create a subprogram or package to avoid writing it a third time.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Thu, 2 Jun 2022 07:56:53 +0200

> What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement?

I would try to fix the problem at where it is caused: ad hoc, unnamed logical predicates! Syntactic sugar won't make these go away.

All those Boolean expressions have meaning, I suppose. The meanings could be given a name. There would be facts, about A, B and C, that make your statement true, some not. What does it state?

Compare this assembly of variables

((A and B) or else ((not A) and C)))

to a lambda expression or to a state machine. Similar? It is the lowest level of computation using a high level language.

> Again, I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search.

Again, there is an algorithm, typically Find_the_First, that will return an index (or cursor). I'd use the return value in a conditional.

From: Brad Moore <bj.mooremr@gmail.com>
Date: Fri, 10 Jun 2022 09:38:35 -0700

> if (A and B) or else (not A and C) then...

> if (if A then B else C) then...

I agree with the other comments, and in a case like this, I might consider writing an expression function to improve readability.

Using cryptic letters for Booleans makes it difficult to assign a name to the expression function, but if you apply it to a less generic example, this becomes easier to do.

For example, if A is renamed to Weekday, B means (time < 9:00pm), and C means (time < 6:00pm) you could write:

```
function Shopping_Mall_is_Open return Boolean is (if Weekday then Earlier_than_9_PM else Earlier_than_6_PM);
```

Then your other code would simply be,

```
if Shopping_Mall_is_Open then ...
```

Brad

Problems Using Generic_Dispatching_Constructor

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Subject: Problems using Generic_Dispatching_Constructor
Date: Wed, 1 Jun 2022 04:36:02 -0700
Newsgroups: comp.lang.ada

The generic function Ada.Tags.Generic_Dispatching_Constructor is defined as:

```
generic
   type T (<>) is abstract tagged limited private;
   type Parameters (<>) is limited private;
   with function Constructor (Params: not null access Parameters) return T is abstract;
function
Ada.Tags.Generic_Dispatching_Constructor (The_Tag : Tag; Params  : not null access Parameters) return T'Class;
```

This gives us some problems when calling an instance of Ada.Tags.Generic_Dispatching_Constructor when the Params parameter is an in-mode parameter of a function e.g.:

```
function Make (From_Params : in P) return T'Class
   is
      function Make_T_Class is new
      Ada.Tags.Ada.Tags.
      Generic_Dispatching_Constructor
      (T => T, Parameters => P,
       Constructor => ...);
```

```
begin
  ...
  return Make_T_Class
         (Some_Tag, P'Access);
  end Make;
```

This results in a compile-time error:

error: access-to-variable designates constant

Why is function Ada.Tags. Generic_Dispatching_Constructor defined as:

```
function
Ada.Tags.Generic_Dispatching_Constructor
(The_Tag : Tag; Params  : not null access
Parameters) return T'Class;
```

and not as e.g (note the access-to-constant type):

```
function
Ada.Tags.Generic_Dispatching_Constructor
(The_Tag : Tag; Params  : not null access
constant Parameters) return T'Class;
```

I guess we could declare function Make as (note the in-out mode):

```
function Make (From_Params : in out P)
return T'Class
```

But this is horrible as functions should never ever have in-out or out-mode parameters (or side effects in general).

Why are access types used at all?

Is there another workaround?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 1 Jun 2022 14:42:40 +0200

> Why are access types used at all?

Parameters are kind of a factory object, you want to have the factory mutable.

> Is there another workaround?

In my practice I never had a case when I could obtain the tag needed for generic dispatching constructor. All my designs ended up with a mapping

key -> constructing function

with an explicit registering the type in the mapping.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 1 Jun 2022 16:25:16 -0500

>> Why are access types used at all?

We needed this usable to implement dispatching stream attributes (the generic dispatching constructor was intended to be a user-definable generalization of the mechanism of the class-wide stream attribute). The stream attributes probably used access types because "in out" parameters were not allowed for functions when they were invented. (So mistakes piled on mistakes. :-)

> Parameters are kind of a factory object, you want to have the factory mutable.

Right. For instance, consider a factory where each object gets a unique id while being constructed. You would want to update the Next_Id component at the end of each construction.

>> Is there another workaround?

> In my practice I never had a case when I could obtain the tag needed for generic dispatching constructor. All my designs ended up with a mapping

> key -> constructing function

>

> with an explicit registering the type in the mapping.

Right. Generally, one uses a mapping of some sort of key or menu choice or whatever to tags. If you aren't adverse to a giant case statement, then you might as well call the constructor directly. (And if you are willing to use access-to-functions, you don't need OOP at all.) So this "factory" is mostly a bone for OOP purists.

The one exception is the case where you have an external tag as the key, since you can get the tag from that directly. But even that is really a mapping (one built by the implementation).