ADA USER JOURNAL

Volume 43
Number 1
March 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,

As I write this preface, we are in between two big Ada events: the FOSDEM Ada Developers Room, which brings together open source enthusiasts presenting their latest developments [1], and the Ada-Europe Int. Conf. on Reliable Software Technologies (AEiC 2022), which this year will return, fingers crossed, as an in-person event at Ghent, Belgium [2]. Information about both can be found in the "Ada-related Events" section.

A piece of news that has made some ripples in the Ada community is the recently announced collaboration between AdaCore and Ferrous Systems to provide a safety-qualified Rust toolchain. The newsgroup saw some reactions to this announcement [3], and a discussion about the merits, similarities and differences between Rust and Ada and their respective strong points.

Glad tidings come for macOS users with the announcement of builds of GCC 12 and SPARK 2014 for this operating system, thanks to the volunteer efforts of Simon Wright [4], with GCC also available for the M1 architecture. And for the lovers of space, some of us wonder whether there is some Ada in the Webb telescope [5]. (Spoiler: probably not.)

Sincerely,
Alejandro R. Mosteo.

[1] "Ada Developer Room at FOSDEM 2022", in Ada-related Events.

[2] "CfC Ada-Europe 2022 Conference", in Ada-related Events.

[3] "AdaCore Joins with Ferrous Systems to Support Rust", in Ada and Other Languages.

[4] "macOS GCC 12.0.1, SPARK2014", in Ada-related Resources.

[5] "Ada in James Webb Space Telescope?", in Ada Inside.

Ada-related Events

CfC Ada-Europe 2022 Conference

[Deadline is past; announcement kept for the record. –arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC Ada-Europe 2022 Conference - 27 Feb - second deadline
Date: Mon, 31 Jan 2022 16:44:08 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

UPDATED Call for Contributions

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 and the Ada Resource Association (ARA)

* 2nd DEADLINE 27 February 2022 *

#AEiC2022 #AdaEurope #AdaProgramming

General Information

The 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022) will take place in Ghent, Belgium, in the week of 14-17 June, in dual mode, with a solid core of in-presence activities accompanied by digital support for remote participation. The conference schedule comprises a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

Schedule

16 January 2022 Submission deadline for journal-track papers, tutorials and workshop proposals.

27 February 2022: Submission deadline for industrial-track and work-in-progress-track abstracts.

14 March 2022 Notification of invitations-to-present for journal-track papers. Notification of acceptance for all other types of submission.

3 April 2022: Publication of advance program.

Topics

The conference is an established international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of developing, running and maintaining challenging long-lived, high-quality software systems for a variety of application domains including manufacturing, robotics, avionics, space, health care, transportation, cloud environments, smart energy, serious games. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software technologies.

The topics of interest for the conference include but are not limited to:

- Real-Time and Safety-Critical Systems: design, implementation and verification challenges, novel approaches, e.g., Mixed-Criticality Systems, novel scheduling algorithms, novel design and analysis methods;
- High-Integrity Systems and Reliability: theory and practice of High-Integrity Systems, languages vulnerabilities and countermeasures, architecture-centred development methods and tools;
- Reliability-oriented Programming Languages (not limited to Ada): compilation and runtime challenges, language profiles, use cases and experience reports, language education and training initiatives;

- Experience Reports: case studies, lessons learned, and comparative assessments.

Refer to the conference website for the full list of topics.

Call for Journal-track Submissions

Following the journal-first model inaugurated in 2019, the conference includes a journal-track that seeks original and high-quality submissions that describe mature research work in the scope of the conference. Accepted papers for this track will be published in the "Reliable Software Technologies (AEiC2022)"

[Submission details removed. Call is closed now.]

Authors who have successfully passed the first round of review will be invited to present their work at the conference. Ada-Europe, the main conference sponsor, will cover the Open Access fees for the first four papers to gain final acceptance, which do not already enjoy OA from personalized bilateral agreements with the Publisher.

Call for Industrial-track Submissions

The conference seeks industrial practitioner presentations that deliver insight on the challenges of developing reliable software. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer review process. Interested authors shall submit a short (one-to-two pages) abstract, by 27 February 2022, via https://easychair.org/conferences/?conf=aeic2022, strictly in PDF, following the Ada User Journal style (cf. http://www.ada-europe.org/auj/).

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be invited to expand their contributions into full-fledged articles for publication in the Ada User Journal, which will form the proceedings of the Industrial track of the Conference.

Prospective authors may direct all enquiries regarding this track to the corresponding chair, Alejandro R. Mosteo, at the listed address.

Call for Work-in-Progress-track Submissions

The Work-in-Progress track seeks two kinds of submissions: (a) ongoing research, and (b) early-stage ideas. Ongoing research submissions are 4-page papers that describe research results that are not mature enough to be submitted to the journal track as yet. Early-stage ideas, are 1-page papers that pitch new research directions that fall in the scope of the conference. Both kinds of submission must be original and shall undergo anonymous peer review. Submissions by recent MSc graduates and PhD students are especially sought.

[Submission details removed. Call is closed now.]

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be offered the opportunity to expand their contributions into 4-page articles for publication in the Ada User Journal, which will form the proceedings of the WiP track of the Conference.

Academic Listing

The Journal of Systems Architecture, publication venue of the journal-track proceedings of the conference, was ranked Q1 (SJR) in the year 2020, also featuring 72th percentile in CiteScope (Scopus). The Ada User Journal, venue of all other technical proceedings of the conference, is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Awards

Ada-Europe will offer an honorary award for the best technical presentation, to be announced in the closing session of the conference.

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic or practitioner slant, including hands-on or practical elements.

[Submission details removed. Call is closed now.]

The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference proper.

[Submission details removed. Call is closed now.]

Call for Exhibitors

The conference will include a vendor and technology exhibition. Interested providers should direct inquiries to the Exhibition Chair.

Venue

The conference will take place in the heart of the city of Ghent, Belgium, capital of the East Flanders province, a halfhour train ride north-west of Brussels. Ghent is rich in history, culture and higher-education, with a top-100 university founded in 1817.

Organizing Committee

* Conference Chair
Tullio Vardanega, University of Padua, Italy
tullio.vardanega at unipd.it

* Journal-track Chair
Jérôme Hugues, Carnegie Mellon University, USA
jjhugues at sei.cmu.edu

* Industrial-track Chair
Alejandro R.Mosteo, Centro Universitario de la Defensa, Zaragoza, Spain
amosteo at unizar.es

* Work-in-Progress-track Chair
Frank Singhoff, University of Brest, France
frank.singhoff at univ-brest.fr

* Tutorial and Workshop Chair
Aurora Agar Armario, NATO, the Netherlands
aurora.agar at ncia.nato.int

* Exhibition & Sponsorship Chair
Ahlan Marriott, White Elephant GmbH, Switzerland
software at white-elephant.ch

* Publicity Chair
Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
dirk.craeynest at cs.kuleuven.be

* Local Chair
Vicky Wandels, University of Ghent, Belgium
Vicky.Wandels at UGent.be

*** Previous Editions

Ada-Europe organizes annual international conferences since the early 80's. This is the 26th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05), Porto, Portugal ('06), Geneva, Switzerland ('07), Venice, Italy ('08), Brest, France ('09), Valencia, Spain ('10), Edinburgh, UK ('11), Stockholm, Sweden ('12), Berlin, Germany ('13), Paris, France ('14), Madrid, Spain ('15), Pisa, Italy ('16), Vienna, Austria ('17), Lisbon, Portugal ('18), Warsaw, Poland ('19), and online from Santander, Spain ('21).

Information on previous editions of the conference can be found at http://www.ada-europe.org/confs/ae.

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

Ada Developer Room at FOSDEM 2022

[Past event, for the record. –arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada Developer Room at FOSDEM 2022 - Sun 6 Feb - online
Date: Thu, 3 Feb 2022 20:14:24 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Call for Participation

11th Ada Developer Room at FOSDEM 2022

Sunday 6 February 2022, online from Brussels, Belgium

Organized in cooperation with Ada-Belgium [1] and Ada-Europe [2]

fosdem.org/2022/schedule/track/ada/

www.cs.kuleuven.be/~dirk/ada-belgium/events/22/220206-fosdem.html

#AdaFOSDEM #AdaDevRoom #AdaProgramming #AdaBelgium #AdaEurope #FOSDEM2022

FOSDEM [3], the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. The 2022 edition takes place on Saturday 5 and Sunday 6 February. It is free to attend and no registration is necessary. This year, for obvious reasons, it has been turned into an online event, just like last year.

In this edition, the Ada FOSDEM community organizes once more 8 hours of presentations related to Ada and Free or Open Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2022 is held on the 2nd day of the event, and offers introductory presentations on the Ada programming language, as well as more specialised presentations on focused topics, tools and projects: a total of 13 Ada-related presentations by 12 authors from 8 countries!

Program overview:

- Introduction to the Ada DevRoom, by Fernando Oleo Blanco, Germany
- Introduction to Ada for Beginning and Experienced Programmers, by Jean-Pierre Rosen, France
- Ada Looks Good, Now Program a Game Without Knowing Anything, by Stefan Hild, Germany
- The Ada Numerics Model, by Jean-Pierre Rosen, France
- 2022 Alire Update, by Fabien Chouteau, France, Alejandro Mosteo, Spain
- SweetAda: Lightweight Development Framework for Ada-based Software Systems, by Gabriele Galeotti, Italy
- Use (and Abuse?) of Ada 2022 Features to Design a JSON-like Data Structure, by Alejandro Mosteo, Spain
- Getting Started with AdaWebPack, by Max Reznik, Ukraine
- Overview of Ada GUI, by Jeffrey Carter, Belgium
- SPARKNaCl: a Verified, Fast Re-implementation of TweetNaCl, by Roderick Chapman, UK
- The Outsider's Guide to Ada: Lessons from Learning Ada in 2021, by Paul Jarrett, USA
- Proving the Correctness of the GNAT Light Runtime Library, by Yannick Moy, France
- Implementing a Build Manager in Ada, by Stephane Carrez, France
- Exporting Ada Software to Python and Julia, by Jan Verschelde, USA
- Closing of the Ada DevRoom, by Dirk Craeynest, Belgium, Fernando Oleo Blanco, Germany

The Ada at FOSDEM 2022 web-page will have all details, such as the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info, including live video streaming and chat, plus recordings afterwards. For the latest information at any time, contact Fernando Oleo Blanco <irvise@irvise.xyz>, or see:

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/

[2] http://www.ada-europe.org/

[3] https://fosdem.org/2022/

Dirk Craeynest, FOSDEM Ada DevRoom team
Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Ada Developer Room Videos Online

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada Developer Room at FOSDEM 2022 - videos online
Date: Sun, 20 Feb 2022 14:23:10 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

** Presentations and video recordings available online ***

11th Ada Developer Room at FOSDEM 2022

held on Sunday 6 February 2022, online from Brussels, Belgium

https://fosdem.org/2022/schedule/track/ada/

All presentations and video recordings from the 11th Ada Developer Room, held at the online FOSDEM 2022 event recently, are available.

Yet another full day with 13 Ada-related talks by 12 authors from 8 countries!

[See program overview in the previous message. –arm]

Thanks once more to all presenters and helpers for their work and collaboration, thanks to Fer for coordinating the DevRoom, thanks to all the FOSDEM organizers and volunteers, thanks to the many participants for their interest, and thanks to everyone for another nice experience!

#AdaFOSDEM #AdaDevRoom #AdaProgramming #AdaBelgium #AdaEurope #FOSDEM2022

Dirk Craeynest, FOSDEM Ada DevRoom team

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Ada-related Resources

[Delta counts are from Nov 1st to May 9th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Mon, 2 May 2022 11:39:21 CET
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_302 (+88) members [1]
- Reddit: 8_005 (+357) members [2]
- Stack Overflow: 2_212 (+87) questions [3]

- Libera.Chat:75 (=) concurrent users [4]

- Gitter:115 (+24) people [5]

- Telegram: 139 (+9) users [6]

- Twitter: 30 (-197) tweeters [7]

53 (-223) 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: Mon, 9 May 2021 11:45:21 CET
To: Ada User Journal readership

Rosetta Code: 900 (+54) examples [1]

39 (+1) developers [2]

GitHub: 763* (=) developers [3]

Sourceforge: 274 (+1) projects [4]

Open Hub: 214 (=) projects [5]

Alire: 243 (+48) crates [6]

Bitbucket: 88 (=) 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: Mon, 9 May 2021 11:50:21 +0100
To: Ada User Journal readership

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

- TIOBE Index: 27 (+4) 0.46% (+0.04%) [1]

- PYPL Index: 17 (=) 0.81% (-0.13%) [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/

Ada "Coin" Updated for Ada 2022

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada - In Strong Typing We Trust - "coin" updated for Ada 2022
Date: Sat, 5 Feb 2022 16:04:59 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Ada - In Strong Typing We Trust - "coin" updated for Ada 2022

As of today, a new version of the traditional "Ada coin" is available for promotional use at http://www.cs.kuleuven.be/~dirk/ada-belgium/pictures/ada-strong.html

Coinciding with the final stages in the ISO standardization of the latest Ada programming language revision, referred to as "Ada 2022", and for the occasion of the 11th Ada Developer Room at FOSDEM 2022, a new update was made available, adding "2022".

Enjoy!

Dirk Craeynest

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/-Europe/SIGAda/WG9 mail)

New Ada Forge

From: William <william@sterna.io>
Subject: New Ada Forge: catalog of (almost) all Ada open source code & tools
Date: Sat, 12 Feb 2022 19:47:36 +0100
Newsgroups: comp.lang.ada

Hello Ada lovers!

I've the pleasure to announce a ground-up update of AdaForge.org

https://www.adaforge.org

The purpose of this site is to bring to the Ada developer a catalog of (almost) all Ada open source code and tools existing in different public repositories.

==> This catalog is structured according to a software developer perspective (taxonomy).

Note: AdaForge.org references 100% of the Alire 'crates' packaging repo. :-)

I'm excited to hear some feedback from you,

Kind regards,
William

Ada-related Tools

AdaStudio-2021 Release 01/10/2021 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce: AdaStudio-2021 release 01/10/2021 free edition
Date: Fri, 1 Oct 2021 22:31:52 -0700
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2021 new release, based on Qt-6.2.0-everywhere Qt 6.2.0 opensourc without qtwebengine,extended with modules from Qt-5.15: qtgraphicaleffect qtlocatio qtgamepad qtspeech qtx11extras qtwinextras Qt 6 is a new long time project and I hope to add qtwebengine in next releases.

Qt6ada version 6.2.0 open source and qt6base.dll ,qt6ext.dll (win64),libqt6base.so,libqt6txt.so(x86-64) built with Microsoft Visual Studio 2019 x64 Windows, gcc x86-64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux x86-64 Debian 10.4 Qt-6.2.0 everywhere opensource prebuilt binaries for win64 and amd64 are included into AdaStudio-2021

AdaStudio-2021 includes the following modules: qt6ada, vtkada, qt6avada, qt6cvada and voice recognizer.

Qt6Ada is built under GNU GPLv3 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/folderview?id=0B2QuZLoe-yiPbmNQRl83M1dTRVE&usp=sharing

[List of detailed file contents omitted. —arm]

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

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Sat, 2 Oct 2021 16:00:50 +0200

Hi Leonid,

I have been following your work for a few years. I like the Qt ecosystem (even with their change of heart) and very specially VTK. Thank you for your work. I hope to use it in the future for my projects.

I first wanted to say that the webpage that is indicated on your CV and where QtAda has been living is unreachable. Google says it has been blocked since it is suspicious. Do you receive the same message? [https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/adastudio.html]

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

Is it GPLv3 or LGPLv3? I am asking since you mention GPLv3 but link LGPLv3.

Once again, thank you for maintaining this lovely software suite!

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 7 Oct 2021 22:59:20 -0700

Qt6Ada is built under GNU LGPLv3 license, sorry for my mistake.

I built a web page from my google drive and it worked well, but now I have got a message from Google and I don't know why. Old link to AdaStudio no longer works. The new is https://drive.google.com/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 6 Oct 2021 23:28:33 +0200

I have been playing today with qt5ada and I think I can shed some light on the issue [of some seemingly missing C files —arm].

The Ada sources that call the Qt procedures are in the AdaStudio/qt5ada/qt5adasrc.tar.bz2 file. That is the source file. There are no C files there. That library contains all of the Ada wrapper.

However, that is indeed not enough to use it. It requires a fully functional Qt5 installation (and a very complete one, with bells and whistles). The binaries are provided in the other *.tar.bz2 files (except the demos file). There is also the qt5adax86-64.tar.bz2 file which weighs about 6Mb. That seems to be the relevant file to build qt5ada from source in Linux. It comes with different files to setup the file structure and environment. I must admit, I have not tried to build it with the provided files in qt5adax86-64.tar.bz2

These "build files" expect you to have a Qt5 installation in your local /usr/local folder. I suppose that is where the qt5.15x86-64.tar.bz2 comes into place, after all, it should unpack in the directory written in the environment file.

Of course, the question is: where are the instructions to build this all from source? The short answer is in the document "How to use Qt5Ada.docx" that is present in AdaStudio/qt5ada. That sheds more light into the procedure. But it still expects you to use the precompiled Qt5 binary. And, I must be honest, it is not clear and easy to follow, you need to adapt the generic instructions to what is on your system...

Then the question becomes: "How can I build _everything_ from source? Specially with the system provided libraries, such as the system provided Qt5." Well... That is not so simple. I understand why Leonid has set up things this way. Correctly setting the compiler flags and directories for system installed libraries is a nightmare. I tried to compile qt5ada with my system provided Qt5 (OpenSUSE Tumbleweed), it is not trivial _at all_. There can be problems with the Qt5 version, there can be problems with the plugins, compiler flags, etc. Can it be done? Most likely, but it will require some elbow grease. There is a reason to why most Qt projects use CMAKE to build and link themselves; because it is not an easy task.

So I would say that the instructions need to be cleaner and that in its current state, there is only one easy solution to building qt5ada, and it requires the binaries provided. But I would also say that all the source files needed are in there. The prebuilt Qt5 binary seems to be the standard unmodified Qt5 distribution, so no surprises there. And that a lot of extra work would be needed to make qt5ada work seamlessly with the system provided libraries.

SweetAda 0.9

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda 0.9 released
Date: Sun, 3 Oct 2021 06:32:40 -0700
Newsgroups: comp.lang.ada

I've just released SweetAda 0.9.

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 (delayed)

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

Clone repository @ https://github.com/gabriele-galeotti/SweetAda

Release notes

There are too many changes, so I will list only the most important features of this release.

- Windows environment does not need the grep utility, nor a dos2unix utility (which is now provided internally); elftool is now optionals and its use is configurable in configuration.in
- RTS can be build from sources by means of "CPU=<cpu> make rts" command (the RTS type is being picked up from configuration.in as usual), every RTS branch will be named like the toolchain triplet being used
- Both SweetAda and RTS are fully buildable in Linux, Windows/cmd.exe, Windows/MSYS and OS X; you should only to have online a "make" and "sed" (and for Windows these are available as zip packages in Sourceforge); due to this, there are no RTS packages anymore
- SweetAda does not relies on SweetAda toolchains, you can use your own GNU toolchain, or whatever GNAT you can pick, just be sure to use Ada 2020

The final result is a package that is fully auto-consistent, because the core, RTS and utilities are fully provided in both source form and executable form. Since SweetAda toolchains are by no means eligible as the unique compilers for the system, they will slowly fade away.

GCC Release Notes, aka, Ada Is Still Alive!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: GCC release notes, aka, Ada is still alive!
Date: Mon, 11 Oct 2021 20:41:18 +0200
Newsgroups: comp.lang.ada

Hi everybody,

I have been meaning to write this message for a long while, so here it goes.

Reading Phoronix [1] for years, I noticed that with every new GCC release, the biggest changes to GCC and its languages were mentioned. However, Ada was pretty much never present.

Today, just a few moments ago in #netbsd, someone asked whether Ada had finally been dropped out of GCC... I am not even mad. GCC's release notes have not mentioned Ada since GCC 8 [2], [3], [4]; and even in GCC 7 and 8 the notes are minute.

So I would like to ask whether someone would like to help me get release notes ready. I am not saying that I will be doing

much, but I would like to breathe some fresh air into how Ada is seen and how much people hear about it.

I personally do not like marketing since good products stand on their merits, not slogans or shininess. But there is no reason to not put publicly what is going on.

Yes, AdaCore has been doing some very nice followups to the development of Ada in their blog [5]. But the people that go there are already aware of Ada. And since AdaCore is phasing out their GNAT CE system in favour of FSF builds (included in Alire), the relevance of GCC's releases grows.

Note, I am not implying that AdaCore should write the releases. They are doing the bulk of work in GNAT, so I do not think they _need_ to do more. Personally I am glad with what they are doing, but of course, they can write the releases if they so want.

I am especially saddened by the fact that GCC has gotten a substantial amount of support for Ada 2022 and it is not even mentioned. No wonder why people think Ada is dead!

So, if you have any recommendation, or would like to help, then you are more than welcomed!

P.S.: I am already doing my part GNAT in NetBSD x86_64 is working! It has 9 failed ACATS tests, but they are minor. A thousand thanks go to J. Marino and Tobiasu for their enormous help in #ada. Today I will see if I can compile it for armv6 and run it on my RPi!

[1] https://www.phoronix.com/scan.php?page=home

[2] https://gcc.gnu.org/gcc-7/changes.html

[3] https://gcc.gnu.org/gcc-8/changes.html

[4] https://gcc.gnu.org/gcc-9/changes.html

[5] https://blog.adacore.com/

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 12 Oct 2021 05:54:33 -0700

Most, if not all, of what is in this blog post [1] is applicable to GNAT/GCC 11.

[1] https://blog.adacore.com/ada-202x-support-in-gnat

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 12 Oct 2021 21:37:25 +0300

> Most, if not all, of what is in this blog post [1] is applicable to GNAT/GCC 11.

I guess the main point of Fernando was that it would be nice if someone could add all the new changes between versions 11 and 12 to https://gcc.gnu.org/gcc-12/changes.html before GCC 12 is released.

gcc-X/changes.html traditionally lists some items for all other language frontends, but there is never anything for Ada.

The git history for gcc-12/changes.html page is visible at

https://gcc.gnu.org/git/?p=gcc-wwwdocs.git;a=history;f=htdocs/gcc-12/changes.html;h=f38fd2bef9c4089369e6f9315590ebffd8b24f5c;hb=HEAD

(that is gcc-wwwdocs repository at gcc.gnu.org/git).

Maybe someone with enough free time (and enough knowledge about the changes) could take look and provide a patch for GCC web page maintainers?

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 13 Oct 2021 18:32:30 +0200

Thank you to everybody that commented on the topic.

We could use the Changelog present in the gcc/ada directory to triage commits more precisely (credit goes to Stéphane).

> I decided to try an example. I must confess that I don't know where the cutoff point for GCC 11 was and what it changes actually did

To be honest, we could try to write the changelog for GCC 11 with the information given by Fabien (AdaCore) and what we find out. If for whatever reason the GCC people do not want to make large changes to the already released changelog, we could compile a larger list for GCC 12.

I think the most important aspects are:

- Ada 2022, which has a long list of changes on its own;
- Improvements to systems (VxWorks, RTMS, etc), as it shows that Ada is present in more places than what meets the eye;
- Deprecations and fixes;
- General improvements in the library, SPARK and with the GCC ecosystem.

I think Ada has somewhat acceptable support for OpenMP, which was improved in the past few years, for example. It has also been increasing SPARK support in the libraries.

[...]

I want to sign up for GCC's gcc mailing list (general discussion) and ask the GCC people what would be the preferred way to move forward. Hey, maybe they would like to have Ada changelogs for all past releases! If I hear anything back I will tell you.

Though if someone wants to start, I see no problem on sharing diffs here. Not the most ideal place, but it is a good forum to share ideas.

From: Simon Wright <simon@pushface.org>
Date: Wed, 13 Oct 2021 20:59:47 +0100

> - General improvements in the library, SPARK and with the GCC ecosystem.

Not sure how to work SPARK into a GCC note, since it's not part of the GCC ecosystem?

"There's extensive support for possible static analysis of code, e.g. via SPARK, in the form of annotations that can optionally be compiled as runtime assertions."

From: Stéphane Rivière <stef@genesix.org>
Date: Thu, 14 Oct 2021 10:24:20 +0200

> So, if you have any recommendation, or would like to help, then you are more than welcomed!

I second that and I would like to help, if I may.

According to gcc-mirror on github, Ada basecode is above C++

C 47.7%
Ada 17.5%
C++ 14.9%
Go 7.4%
GCC Machine Description 4.7%
Fortran 2.4%
Other 5.4%

git clone https://github.com/gcc-mirror/gcc
git log > log.gcc (volume: 124M)
cat log.gcc | grep AdaCore > log.ada (1M, ~25K contribs since 2005)
grep "\[Ada\]" ./log.gcc > log-oneliner.ada (190K, 3200 lines)
grep -B 2 -A 20 AdaCore log.gcc > log-detail.ada
cat log-detail.ada | grep -B 2 -A 20 [ada] log.gcc > log-changes.ada

It seems that everything is there to create a more or less relevant changelog.

But AdaCore's comments are one thing, sorted and relevant information for developers are another.

A raw copy/paste would be useless, we would have to analyze the changelog to give back useful information.

We should also edit a changelog for each GCC release. The above metrics were made on master.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 10:42:20 +0200

The discussion thread on the GCC ML has been started. You can find it here: https://gcc.gnu.org/pipermail/gcc/2021-October/237600.html

Do not hesitate to add any comments!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 22:14:57 +0200

> The discussion thread on the GCC ML has been started.

Okay, we already had a couple of comments and they cover everything needed. Arnaud has volunteered to be the "supervisor". So here is my plan: crowdsourcing! :D

I would like to write a (simple) list of changes for each version here, on the CLA. If you want to add something __copy__ (do not quote) the list from the previous person/reply/modification and add your proposed changes. You can also make comments if you would like anything changed. If "CHECK" or if "TODO" are written by somebody, it means that something needs to be checked or that it needs to be expanded; respectively. After the list is mostly completed, we could create a patch(es) to send to GCC. The quality of this list is not going to be great, treat it like a checklist. Obviously, if you want to discuss something about the changes, do quote the relevant section.

[...]

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 25 Oct 2021 20:47:05 +0200

Diff: add to GCC 12 the deletion of gnatxref and gnatfind (the patch was posted today in the ML). The -gnat2020 has been commented too in GCC 10 and -gnat2022 in GCC 12. Also, we have explicit permission by Arnaud to copy as much code as necessary from AdaCore's blog.

LIST OF CHANGES

GCC 12

- Introduction of the -gnat2022 flag in gnatmake (-gnat2020 is a deprecated alias).
- gnatfind and gnatxref tools have been deleted. They have been deprecated for years and have been substituted by gprbuild tools.
- Further library improvements in both quality and performance.
- The use of contracts has been extended in the "Ada library" allowing for further checks at runtime or a deeper static analysis with the SPARK prover.
- Further improvements to embedded systems such as VxWorks and RTMS. CHECK maybe be more specific/generic.

GCC 11

- Better Ada 2022 support. The parallel keyword is still unsupported.
- TODO name the additional features. See [1], obviously, with some code examples.
- Addition of the Jorvik profile. CHECK, see [2], maybe code examples?
- Additional non-standard features [3]. CHECK if this applies to GCC 11 or 12.
- A bug was fixed were previous GCC versions allowed XXX construct CHECK. This is not allowed by the standard. Some software was making use of XXX (which is, once again, not allowed) and it has to be patched.
- General library improvements in both clarity and performance.
- The use of contracts has been extended in the "Ada library" allowing for further checks at runtime or a deeper static analysis with the SPARK prover.
- Further improvements to embedded systems such as VxWorks and RTMS. CHECK maybe be more specific/generic.

GCC 10

- Introduction of the -gnat2020 flag in gnatmake (-gnat2020 is a deprecated alias). It enables newer features present in Ada 2022 (still to be ratified). These features are still experimental.
- Some Ada 2022 features are available already with the use of the -gnatX (gnat eXtensions switch).

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 27 Oct 2021 09:52:26 -0700

> - gnatfind and gnatxref tools have been deleted. They have been deprecated for years and have been substituted by gprbuild tools.

What "gprbuild tool" replaces gnatxref?

From recent discussions in an AdaCore ticket, the replacement for gnatxref is libadalang, either via the LSP Ada Language Server, or a similar custom wrapper.

GCC Updated in NetBSD!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: GCC updated in NetBSD!
Date: Tue, 19 Oct 2021 23:47:36 +0200
Newsgroups: comp.lang.ada

Hello everybody! I bring good news!

GCC with Ada support has been updated in NetBSD! Now versions 10 and 11 should work on x86 and x86_64 NetBSD machines! You can find them in pkgsrc-wip (gcc10-aux) [1] and Ravenports (gcc11) [http://www.ravenports.com/]!

First things first, the acknowledgements: a big thank you goes to J. Marino who did the original gcc-aux packages and who provided most if not all the work when it came to fixing the threads and symbols. Another big thank you goes to tobiasu who correctly picked up that the pthread structure wrappers were not correct and had to be remade. Another big thank you goes to Jay Patelani for his help with pkgsrc.

So, long story short. Most of the work that had been done up until a few weeks ago was done correctly, but the failing tests (most related to tasking) were failing in very strange ways. It happened that the pthread structure memory that the Ada wrapper was using was incorrect, so we were getting completely erratic behaviour. Once that got fixed, pretty much all tests passed. J. Marino also took the time and effort to create __gnat_* function wrappers to all the symbols that the NetBSD people have renamed. This is a much cleaner fix and allows for the renamed functions to generate the correct symbols since now they are getting preprocessed. It should also be more "upstream friendly". The issue, however, remains if NetBSD decides to rename more functions that are still being linked directly.

There are still some failing ACATS tests (about 10). Some are related to numerical precision and a couple others. They are mostly the same failing tests in both GCC 10 and 11. J. Marino ran the ACATS tests on a DragonflyBSD (or was it FreeBSD?) machine and the same tests were failing there too. So we suspect is is a common limitation on *BSDs and it is unlikely that this will ever affect anybody. There is also the issue of stack unwinding when it contains a signal trampoline [2], read the following thread to gain more information about this.

[1] https://github.com/NetBSD/pkgsrc-wip/tree/master/gcc10-aux

[2] https://mail-index.netbsd.org/tech-kern/2021/10/15/msg027703.html

I have started trying to get GCC to xcompile to arm* on NetBSD. I think I am somewhat close, but further hacking on NetBSD's src is needed (and I think the RTS is not getting picked up correctly). So do not get your hopes up. I mean, I have a working gcc x86_64 NetBSD host to NetBSD arm* xcompiler, it is the native gcc on arm* that is not getting built correctly.

From: Richard Iswara <haujekchifan@gmail.com>
Date: Wed, 20 Oct 2021 12:01:40 +0700

A big applause for your hard work identifying the problem in the first place.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Tue, 19 Oct 2021 23:43:23 -0700

When I was working at AdaCore, we used to run our internal CRM and the ticket-management tool that processes all email

on a FreeBSD machine, because the sysadmin was very fond of that system. The CRM was (is?) based on AWS (Ada Web Server), so using tasking pretty heavily. We never had any problem at the time.

I guess AdaCore has given up on FreeBSD, like they have macOS.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 20:44:01 +0200

> I guess AdaCore has given up on FreeBSD, like they have macOS.

Well, GCC officially supports FreeBSD x86* and AFAIK, arm too. Though, AFAIK, the gcc-aux packages from freshports have been left without a maintainer...

And good news everybody! I have managed to get GPRBuild working and Alire too! I even got the GNATColl components built using Alire ^^. Pretty easy if you ask me :P

The mayor issue I am facing now is with make... I tried building AWS with Alire but it could not, since it was using make, which in *BSD world is BSD make, aka, bmake, not GNU make, aka gmake... Anyhow, I am very happy to see so many packages getting built without issues in NetBSD :D

There is the problem where GPRBuild says that the "lib" option is not supported on the OS. I don't think it is suprissing since GPRBuild probably does not know anything about NetBSD.

I am also getting warnings from gnatmake:
/home/fernando/bootstrap_ada/alire/src/alire/alire-toolchains.adb:331:8:

warning: frame size too large for reliable stack checking which probably come from NetBSD having a small stack by default.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 2 Nov 2021 21:32:56 +0100

A bit of a followup.

The package gcc10-aux has been updated in pkgsrc-wip. I am now the maintainer. As requested by some pkgsrc developer, I have made the package explicitly depend on gcc6-aux. That way, it may be used as a base Ada compiler for all the packages that need Ada (although this is just the first step). I have also rebased it on the new skeleton of gcc10 from pkgsrc-current. Hopefully the review period and inclusion into pkgsrc-current will not take much time.

[...]

From: Fernando Oleo Blanco <irvirse_ml@irvise.xyz>
Date: Thu, 23 Dec 2021 12:52:42 +0100

Well well well...

I come with a Christmas present... Ada running on NetBSD-powerpc! It should run on any powerpc "port", in NetBSD terms also known as evbppc, macppc and amigappc.

It is not perfect, but it is there.

Here are the results from ACATS 4.1X running natively on the macppc port (as created by https://github.com/alarixnia/mkimg-netbsd)

=== acats Summary ===

# of expected passes	2490
# of unexpected failures	62
# of expected failures	1487
# of unresolved testcases	11
# of unsupported tests	116

*** FAILURES: c324006 c350a01 c452003 c452005 c452006 c452a02 c52103x c52104x c52104y c552a01 c552a02 c611a04 c650b04 c760a02 c96001a c96008a c96008b cb1010a cb1010c cb1010d cc40001 cc51007 cdd2b03 cdd2b04 cxa4010 cxa4011 cxa4021 cxa4022 cxa4023 cxa4030 cxa4031 cxa4032 cxa4033 cxa4035 cxaa022 cxab004 cxab005 cxac004 cxag001 cxag003 cxai001 cxai009 cxai010 cxaia01 cxaib05 cxaib06 cxaib08 cxb4002 cxb4005 cxb5002 cxb5003 cxd1003 cxd1004 cxd1005 cxd2002 cxd2003 cxd2004 cxd2006 cxd3001 cxd3002 cxd6001 cxd6002

/home/fernando/ACATS-master/run_all.sh completed at Thu Dec 23 10:13:16 UTC 2021

The compiler is GCC from the NetBSD src tree, which is an older GCC 10 version. Which means (following the results from previous runs) that 28 failures where expected; 6 from shortcomings from NetBSD and the rest from GCC 10 not passing newer tests. That means this system generated at least 34 new failures. This may be for a number of reasons, both related and unrelated to GCC-Ada. Still, I think they are rather good! I believe a lot of cxa failures were due to the system running on low memory. Also, the compiler was built against NetBSD 9.99.92, but the actual host is 9.2, and NetBSD is not backwards compatible; so that may explain other failures.

Just for your own enjoyment, these tests took about 2 days to run, since I am emulating powerpc on a virtualised NetBSD-x86_64 system :P

The reason I tried to run powerpc is because, to put it bluntly, NetBSD has to fix their shit with aarch64 and mips64 and because they do not provide binaries for POWER. NetBSD just works if you use their tooling, but the moment something out of the ordinary of what has to be built, fecal matter impacts the air impeller (credit to a reddit user for that one).

Merry Christmas everybody!
Fer

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 10 Feb 2022 20:21:53 +0100

One last update on GCC 10 on NetBSD.

As I have already said in other messages, it works great. The package is still under wip since no maintainer has stepped up to take care of the review. I also have not pushed it further.

I would recommend the use of Ravenports, since it has GCC 11, which is newer and works on FreeBSD too.

I have given up on trying to port it to other arches. It should be as simple as adding them to the Makefile.rtl. There is a minor bug on my patchset, the x86 intrinsics are also present on the arm sections, I need to delete that.

The reason for giving up on supporting other arches is mostly due to NetBSD not upstreaming support for those arches. For example, the official binutils does not have support for aarch64-netbsd. It is only present in NetBSD's src. And it only works when used within NetBSD's src. This makes everything more complex than needed and I do not have the will to push through with it.

Regarding the use of other Ada tools in NetBSD. I added support for grpbuild a few months ago, so you should be able to just use it. Notice, when using GCC 10 only V21 of AdaCore tools work. Newer versions (currently v22) need GCC 11. The rest of the tools seem to compile without much fuzz at all. So I say that my work is mostly complete.

I will try to get gcc10-aux pushed to stable however; sometime after March.

For now, I will try to update the Ada changelog in GCC and write an article about Ada-Scheme for the AUJ.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 14 Mar 2022 22:21:49 +0100

Quick update. The package has now been upstreamed and is now part of the official pkgsrc distribution!

You can find it here:
https://cdn.netbsd.org/pub/pkgsrc/current/pkgsrc/lang/gcc10-aux/index.html

Binaries are still not available since it just got added.

This is a nice conclusion to this journey... But there is something else brewing behind the scenes... AVR support is coming to Alire thanks to Fabien and we are ironing out some of the issues there :D

Simple Components v4.59

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada v4.59
Date: Sat, 6 Nov 2021 13:04:27 +0100
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:

- The primitive operation Clear was added to GNAT.Sockets.Server.Connect;
- Julia bindings moved to the version 1.6.3;
- The functions Eval_char_array, Get_Safe_Restore, Load_File_String, Set_Safe_Restore were added to Julia bindings;
- Functions To_Julia and Value added to Julia bindings for Ada types Time and Day_Duration;
- To_Julia defined on tuples fixed when types of elements are not directly visible.

Dokan Ada Bindings 2.0

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Dokan Ada bindings 2.0
Date: Sun, 16 Jan 2022 17:07:54 +0100
Newsgroups: comp.lang.ada

Dokan is a user-space file system for Windows 32- and 64-bit. It consists of a driver and a library. The driver routes the I/O requests to the file system device to the library callback.

A sample implementing a memory resident files system is provided.

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

Changes to the version 1.5.0:

- The code and the API were reworked to accommodate the Dokan major version 2.

AdaControl 1.22r16c

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] New version of AdaControl
Date: Wed, 8 Dec 2021 17:51:44 +0100
Newsgroups: comp.lang.ada

AdaControl 1.22r16c is mainly a bug fix release (no new rule), but improvements in the static evaluator provides better results and avoids false positives in several rules.

Enjoy!

Renaissance-Ada Made Open Source

From: Pierre Van De Laar <pierre.van.de.laar@gmail.com>
Subject: Renaissance-Ada, a toolset for legacy Ada software, made open source
Date: Thu, 27 Jan 2022 04:32:00 -0800
Newsgroups: comp.lang.ada

Dear Members of comp.lang.ada,

We would like to inform you that we have made Renaissance-Ada, a toolset for legacy Ada software, open source:

https://github.com/TNO/Renaissance-Ada

The Renaissance-Ada project builds on top of LibAdalang and includes the following functionality

* Dependency Graph Extractor that produces a graphml file for visualization and querying with e.g. yEd and Neo4J.
* Rejuvenation Library that allow analysis and manipulation of Ada code based on concrete patterns.
* Rewriters_Library that enables automatic rewriting of Ada code based on concrete patterns.
* Code Reviewer that automatically reviews Ada code based on a large list of rewrite rules.

If you have any question about this toolset don’t hesitate to contact me!

GWindows Release, 29-Jan-2022

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: GWindows release, 29-Jan-2022
Date: Sat, 29 Jan 2022 13:48:46 -0800
Newsgroups: comp.lang.ada

GWindows is a full Microsoft Windows Rapid Application Development framework for programming GUIs (Graphical User Interfaces) with Ada. GWindows works only with the GNAT development system, but with some effort, GWindows could be made pure Ada. GWindows is free and open-source!

Changes to the framework are detailed in gwindows/changes.txt or in the News forum on the project site.

In a nutshell (since last announcement here):

427: GWindows.Image_Lists: added color options; includes features of "extended" Ex_Image_List_Type in package GWindows.Image_Lists.Ex_Image_Lists, which is marked as obsolescent.

424: GWindows.Application: added function Screen_Visibility.

423: GWindows.Application: added Enumerate_Display_Monitors.

422: GWindows.Base: added Set_Foreground_Window.

421: GWindows.Base: added Set_Active_Window.

417: GWindows.Common_Controls.Ex_Tb (toolbar): is now 64-bit compatible; see LEA http://l-e-a.sf.net/ , LEA_GWin.Toolbars for an example.

414: GWindows.Scintilla: method names are "de-camel-cased": e.g.: "Move_Caret_Inside_View" instead of "MoveCaretInsideView".

412: GWindows.Scintilla: works on both Intel x86 32-bit and x64 64-bit types of platforms.

411: GWindows.Common_Controls.Ex_List_View: method On_Free_Payload is now public and can be overridden with effect.

410: GWindows.Common_Controls.Ex_List_View: Sort can use a comparison method not based on strings (e.g. a numerical comparison).

GWindows Project site:

https://sf.net/projects/gnavi/

GWindows GitHub clone:

https://github.com/zertovitch/gwindows

Enjoy!

macOS GCC 12.0.1, SPARK2014

From: Simon Wright <simon@pushface.org>
Subject: ANN: macOS GCC 12.0.1, compatible SPARK2014
Date: Fri, 25 Feb 2022 18:21:04 +0000
Newsgroups: comp.lang.ada

GCC 12.0.1 of 20220204 (only Ada, C, C++, built on El Capitan, runs up to Monterey) available at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.0.1.

SPARK2014 built against it (provers CVC4, Z3, Alt-Ergo; CVC4 requires Sierra and upwards) available at

https://github.com/simonjwright/spark2014/releases/tag/macos-0.1.

Needs GCC 12.0.1 installed. Running the test suite on the ug* tests (the examples in the User Guide) results in one failure (aside from the missing CodePeer one) unless you build with -j2 (where 2 is less than the number of processors in your machine).

A note on building the latter at https://forward-in-code.blogspot.com/2022/02/spark2014-and-fsf-gcc.html.

UXStrings Package Available (UXS_20220226)

From: Blady <p.p11@orange.fr>
Subject: [ANN] UXStrings package available (UXS_20220226).
Date: Tue, 1 Mar 2022 21:47:49 +0100
Newsgroups: comp.lang.ada

The objective of UXStrings is Unicode and dynamic length support for strings in Ada.

UXStrings API is inspired from Ada.Strings.Unbounded in order to minimize adaptation work from existing Ada source codes.

Changes from last publication:

- Ada.Strings.UTF_Encoding. Conversions fix is no longer needed with GNAT CE 2021

- A few fix

Available on GitHub https://github.com/Blady-Com/UXStrings and also on Alire https://alire.ada.dev/crates/uxstrings.html

Feedback is welcome on actual use cases.

GCC 12.0.1/Apple Silicon

From: Simon Wright <simon@pushface.org>
Subject: [ANN] GCC 12.0.1/Apple silicon
Date: Wed, 23 Mar 2022 21:08:25 +0000
Newsgroups: comp.lang.ada

Find GCC 12.0.1 and tools for M1 Macs at https://github.com/simonjwright/distributing-gcc/releases/tag/aarch64-apple-darwin21-1

About double the size of the x86_64 (Intel) equivalent.

Ada Inside

Ada in James Webb Space Telescope?

From: Nasser M. Abbasi <nma@12000.org>
Subject: is Ada used in James Webb Space Telescope software?
Date: Sun, 26 Dec 2021 07:18:41 -0600
Newsgroups: comp.lang.ada

Anyone knows if Ada is used in James Webb Space Telescope software.

Either in the control systems or in the embedded software for the Telescope.

https://www.jwst.nasa.gov/

I sure hope they did not use javascript or Python or C for the software.

There is some talk in the following link about its software but I could not find what language they used.

https://www.nasa.gov/feature/goddard/2020/nasa-s-james-webb-space-telescope-completes-comprehensive-systems-test

From: Peter Chapin <peter@pchapin.org>
Date: Thu, 30 Dec 2021 08:30:54 -0500

> Anyone knows if Ada is used in James Webb Space Telescope software.

It is likely they used C. Specifically, C99. I say this because in my dealings with NASA (related to my work with CubeSats), the people I've talked with made it clear that NASA is now a C shop. Both my colleague and I have extolled the virtues of Ada and SPARK to NASA engineers, but we get the usual reaction: too much investment in C to take any other option seriously... except maybe C++ (JPL, at least, does some work with C++ so that might also be on the JWST).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 26 Dec 2021 15:23:43 +0100

Since it was 30 years in development, I would not dismiss QBasic...

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 26 Dec 2021 11:22:56 -0800

> Since it was 30 years in development, I would not dismiss QBasic...

Don't forget Forth! It was used on many space projects.

https://web.archive.org/web/19990125085748/http://forth.gsfc.nasa.gov/

From: John Mccabe <john@mccabe.org.uk>
Date: Sun, 26 Dec 2021 15:57:42 -0800

> Don't forget Forth! It was used on many space projects.

Interesting. I didn't realise there had been so many projects in Forth. I started to learn/use Forth at one point, as it looked like we (Matra Marconi Space) might be forced to use the RTX2010 as it was one of very few space qualified processors with hardware floating point support. In the end we used the MA31750, with Ada, instead.

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 26 Dec 2021 16:37:00 -0800

> I didn't realise there had been so many projects in Forth.

Much of Forth's early development was at the Kitt Peak observatory where I think Charles Moore worked for a while, so it was popular with the astronomy community and maybe indirectly with the spaceflight community through there and JPL. As a more general matter, hardware designers (electrical engineers who sometimes have to muck with embedded software but aren't really into programming as a topic) tend to like it because of its simplicity and directness.

> In the end we used the MA31750, with Ada, instead.

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.

I wonder if targeting GCC to the RTX2010 might have been feasible. Can I ask what Ada compiler you used for the MA31750? It looks like GCC supported the MA31750 until version 3.1, but I don't know whether GNAT existed then.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 27 Dec 2021 09:44:26 +0200

>> I didn't realise there had been so many projects in Forth.

Forth is of course one of the few ways to get a self-hosted but fairly fast interactive compiler/editor system on small processors.

In the 1980's I was working in radio astronomy and we were planning to use Forth to replace HP BASIC on an HP2100 16-bit mini for telescope control and data acquisition. I had a little crush on Forth at the time, but fell out of love with it when I found that some astronomy SW had defined the word 2000.0 as a procedure to convert stellar coordinates to the year 2000 ephemeris... very clear :-(

Fortunately IMO we chose to use HP-Algol instead, and much later changed to Ada on a MicroVAX.

> Can I ask what Ada compiler you used for the MA31750?

Like John, I used Ada on an MA31750. We used the TLD Ada compiler, where (IIRC) TLD stands for the main author, Terry L. Dunbar. GNAT was around, but I don't remember if it had support for the MA31750 -- I doubt it. We used gnatp 3.<something> for testing the MA31750 SW on workstations (Sun Solaris on SPARC, IIRC), but the customer (Matra Marconi Space) specified TLD Ada for the target, so there was never a question of using GNAT instead.

That project developed the on-board SW for the ozone-monitoring instrument GOMOS on the ESA ENVISAT satellite.

I believe ENVISAT used MA31750 and TLD Ada for all its systems.

From: John Mccabe <john@mccabe.org.uk>
Date: Tue, 28 Dec 2021 02:24:54 -0800

> [the MA31750] appears to be a 16 bit processor that implements the MIL-STD-1750A instruction set(!)

There were 3 or 4 different implementations of the MIL-STD-1750A instruction set architecture around the time. It was an interesting one; it was fairly small, but had some relatively complex instructions that were really useful. The MA31750 was GEC-Plessey Semiconductors' 2nd version, I believe, although if I remember correctly, this was the one that had the FPU, or maybe it was the MMU, integrated into a single device, using silicon-on-sapphire for rad-hardness. There were two other implementations I particularly remember that were rad-hard, one by IBM, which had better claimed performance but was really expensive and special order only (I think we paid £7500 or so for each MA31750, so you may be able to imagine what I mean by "really expensive"), and one by another US company that went into Chapter 11 protection around the time we were talking to them!

> Can I ask what Ada compiler you used for the MA31750?

I'm almost 100% sure GNAT wasn't available for the MIL-STD-1750A; it was a very niche market and we weren't aware of any C compilers we could've used at the time, even if we'd wanted to.

The Ada compiler we used was the same as Nikolas; TLD. I was also working on part of ENVISAT (the Tile Control and Interface Unit - TCIU, although some of my colleagues were also using it on the main ASAR control system). Although Nikolas mentions Matra Marconi Space mandating TLD, that would've come down from Dornier who'd apparently done a deal with TLD. I don't know what happened with TLD after that, but some geezer from the Irvine Compiler Corporation contacted me once when they were following up on some unpaid license fees related to part of the TLD compiler.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 28 Dec 2021 12:59:32 +0200

> Although Nikolas mentions Matra Marconi Space mandating TLD, that would've come down from Dornier who'd apparently done a deal with TLD.

Yes, a considerable part of our requirements came from Dornier via Matra Marconi Space (France). We sometimes had fun trying to understand how the French had interpreted requirements written in English by the Germans. The two other languages had left their imprints on the "English" wording :-)

From: John Mccabe <john@mccabe.org.uk>
Date: Fri, 31 Dec 2021 02:26:14 -0800

> Yes, a considerable part of our requirements came from Dornier via Matra Marconi Space (France).

We didn't really have that problem. On TCIU most of our requirements came from Dornier - > MMS-UK (ASAR instrument prime) - > Alcatel - > MMS-UK (TCIU team). Both MMS-UK teams were in Portsmouth. Alcatel were only there because of 'juste retour'* and they didn't even seem to bother trying to interpret the MMS-UK ASAR requirements, they just changed the front page to have "Alcatel" on it. We basically had a shed-load of requirements placed on us that had nothing to do with what the TCIU needed to do, and Alcatel never did get round to formally specifying the bit we really did need from them (the TCIU -> T/R Module - an Alcatel device - interface) as far as I can remember!

It was good in a way, but Alcatel certainly, and possibly also Alenia, played politics all the way through. We were required to go through Alcatel to get them to clarify some of the requirements that were relevant and had come from MMS-UK. As they had no idea what they meant, Alcatel had to go to MMS-UK to get the clarification. Fortunately Alcatel appeared to want to do as little work as possible for their money so they'd just forward the clarification from MMS-UK without bothering to try to understand it.

I'm sure lots of people have been in similar situations, but the inefficiency could've been disastrous, especially as we (the MMS-UK teams) had been working directly with each other on ASAR for years before Alcatel were put in to split us up, and we used the same canteen!

Ah well, those were the days. Apologies for going so far off-topic, but it was nice to reminisce :-)

* Similarly, the ASAR CESA (Central Electronics SubAssembly) requirements came Dornier - > MMS-UK - > Alenia - > MMS-UK.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Subject: [OT] ESA project memories (was Re: is Ada used in James Webb Space Telescope software?)
Date: Fri, 31 Dec 2021 23:18:49 +0200
Newsgroups: comp.lang.ada

> We didn't really have that problem. On TCIU most of our requirements came from Dornier - > MMS-UK (ASAR instrument prime) - > Alcatel - > MMS-UK (TCIU team). Both MMS-UK teams were in Portsmouth.

Interesting :-). I had a similar, but inverse, experience in a later project (SW for the Flexible Combined Imager instruments on the MTG satellites) where Thales Alenia Space (France) was both our customer for the whole SW and our subcontractor for a part of the SW. It led to a number of "direct" communications and decisions between the two TAS-F teams that bypassed our team (in Finland) and of which we learned later. But not much harm done, overall a good project.

> Alcatel were only there because of 'juste retour'*

I can't complain about "juste retour" as without it much less ESA work would be given to Finnish companies, especially earlier when Finland was a new ESA member with no experience in ESA work.

(For those not in the know: "juste retour" is the ESA policy by which ESA tries to give enough project work to each of its member countries to correspond to the country's share of ESA membership fees.)

[...]

> I'm sure lots of people have been in similar situations [...]

Although splitting work up into several companies does easily make for inefficiency, in can also have the benefit of documenting stuff that otherwise might be lost in internal e-mails or face-to-face discussions. That is, if the companies involved do their work properly, and don't act as you describe for Alcatel. But perhaps the Alcatel technical people did as well as they could to mitigate a poor higher-level decision, by being basically a transparent conduit, as you describe.

From: John Mccabe <john@nospam.mccabe.org.uk>
Date: Wed, 5 Jan 2022 16:43:11 -0000

> Interesting :-). I had a similar, but inverse, experience in a later project [...]

It would be inappropriate of me to say whether or not that sort of behaviour occurred on ASAR, although I seem to remember occasions where Alcatel waived their right to be piggy-in-the-middle as some of the discussion about SAR pulse timing and the effect of shifting things around a bit, to deal with the fact that we would've needed a mid-90s supercomputer (and a substantial re-design of the TCIU -> T/R Module interface) to achieve what was originally specified, would've fried the brains of the people who were actually involved :-)

<snip>

> Although splitting work up into several companies does easily make for inefficiency, it can also have the benefit of documenting stuff that otherwise might be lost in internal e-mails or face-to-face discussions. [...]

To be fair (to MMS!), the actual documentation that was produced at the instrument level was pretty good. To be fair to Alcatel, as I mentioned, we'd been working without them on this for a long time before ESA decided to mandate that they should "manage" the TCIU development as a subcontract, so they were forced to pick up on stuff they pretty much hadn't cared about before.

Ironically none of this helped with the documentation from Alcatel; the TCIU -> T/R Module interface I mentioned, for example. We went through 3 rounds of TCIU Software Requirements reviews (i.e. SRR, then re-visited at ADR and DDR or something like that), where our assumptions on how that interface worked (based on rough sketch ideas we'd been given rather than formal specification) were described, before someone at Alcatel bothered to read it and say "nah, doesn't work like that" (presumably in French) :-)

Ada and Other Languages

AdaCore Joins with Ferrous Systems to Support Rust

From: Paul Rubin <no.email@nospam.invalid>
Subject: Adacore joins with Ferrous Systems to support Rust
Date: Wed, 02 Feb 2022 00:57:33 -0800
Newsgroups: comp.lang.ada

https://blog.adacore.com/adacore-and-ferrous-systems-joining-forces-to-support-rust

Ferrous Systems is apparently a Rust support company based in Germany. From the linked page:

"Ferrous Systems and AdaCore are announcing today that they're joining forces to develop Ferrocene - a safety-qualified Rust toolchain, which is aimed at supporting the needs of various regulated markets, such as automotive, avionics, space, and railway."

No mention about whether there will be any type of FOSS or community release. No word on whether the compiler and/or toolchain will be based on the existing stuff, or something new. Wonder how they will safety-certify anything in Rust when the language itself doesn't even have a formal spec. But, it is an interesting development.

Is the writing on the wall for Ada?

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 2 Feb 2022 13:04:42 +0000

I see this going one way, Ada loses out as the Rust side uses AdaCore to get what they want.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Wed, 2 Feb 2022 07:29:12 -0800

If possible please tell what Rust has to offer over Ada. From a quick look at the Rust book it seemed weaker in structured programming, generic programming, type system.

Thanks.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 02 Feb 2022 08:19:37 -0800

> Is the writing on the wall for Ada?

Yes. And it says:

As long as people care about quality software engineering, they will use Ada.

:)

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 2 Feb 2022 16:36:46 +0000

> If possible please tell what Rust has to offer over Ada.

[...] not a lot, only the borrow checker stuff.

From: John Mccabe <john@mccabe.org.uk>
Date: Thu, 3 Feb 2022 15:29:17 -0800

> If possible please tell what Rust has to offer over Ada.

A very nasty syntax, and there's an annoying thing where it moans about what you've called your project.

TBH that's about as far as I got with Rust; I can't be doing with pedantic restrictions that have no technical benefit (as far as I can see).

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Wed, 2 Feb 2022 10:48:44 -0800

> Is the writing on the wall for Ada?

Don't worry too much, people said that already more than 30 years ago... But perhaps the company will rebrand itself RustCore :-) ?

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 12:03:03 -0800

> But perhaps the company will rebrand itself RustCore :-) ?

If the new IDE is called Oxide, watch out ;-).

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 12:06:16 -0800

> I see this going one way, Ada loses out as the Rust side uses AdaCore to get what they want.

I don't think this is two companies merging. It's two companies working jointly on a particular product. Idk any more than the press release though.

Regarding Rust vs Ada, I've never heard anything from anyone who is a real expert at both. Superficially it looks to me like Rust's type system really is more precise than Ada's in general, although it doesn't have integer range types. In other stuff like modules, Ada is probably still ahead.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Wed, 2 Feb 2022 21:07:44 +0100

> If possible please tell what Rust has to offer over Ada.

Embedded systems reportedly still mostly use C. Consequently, C is the thing to which a language's merits must be compared. If Rust managers manage to persuade C shops to use Rust, the this is a reasonable attempt at improving C based program production. If Ada influences the process, then it has had a purpose. :-)

Perhaps it is easier to add guarantees to Rust programs than to C programs. Also, C programmers might feel more at home when using Rust. Java programmers also, even when it is just curly braces.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 3 Feb 2022 01:34:40 +0000

> [...] have integer range types.

Apparently, they have ranges as templates, can't see how that compensates.

[...]

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 18:20:04 -0800

> [Ada vs Rust] Er, try learning both and you'll see?

It's a big effort to become expert at either, let alone both. Basic or superficial knowledge isn't helpful for such comparisons. I've read "Ada Distilled" (Ada 95 version) but still have no clue understanding most of the Ada discussions in this newsgroup, so there is a big gap between basic and advanced knowledge of Ada.

>> [type] system really is more precise than Ada's [...]

From what I've heard, Rust's type system is similar to Haskell's. Haskell's type system can verify stuff just by typechecking, that might be doable in Ada using SPARK and maybe an external proof assistant, but not with just types. Example: a red-black tree using Haskell GADT's (Generalized Algebraic Data Types):

https://www.reddit.com/r/haskell/comments/ti5il/redblack_trees_in_haskell_using_gadts_existential/

> Ada's ahead in most things.

Idk, I'd like to know more. I've never done anything serious in Ada and nothing at all in Rust, but C++ seems a lot more fluid than Ada, and Rust is supposed to compare well with C++. C++ of course is terrible in terms of reliability but I'm only referring to the effort needed to bang out a chunk of code.

From: Luke A. Guest
<laguest@archeia.com>
Date: Thu, 3 Feb 2022 02:52:25 +0000

> It's a big effort to become expert at either, let alone both. Basic or superficial knowledge isn't helpful for such comparisons.

You don't need to learn both languages inside and out. You pick a project and implement it in both languages, that project has to be specific to what you are wanting to know about whether it's binding or tasking or whatever, Ultimately with Ada, you can get by knowing the Ada subset and the representation clauses and do most of what you need to do to compare to another language, obviously if you want to compare tasking, you need to go further.

> [...] there is a big gap between basic and advanced knowledge of Ada.

Learn the basic part, it's equivalent to Pascal with proper type ranges, Pascal only has subtypes iirc. That's the main part of the RM, no annexes, you can do without access and tagged/controlled types to start with. At uni, we didn't even touch on tagged types, but used controlled types, it is possible.

> Haskell's type system can verify stuff just by typechecking

So it's no different than a C++ compiler checking against classes or a variant of C with strong typing? Still no ranges, subranges, ranges with holes in, etc. This is where the power is.

[...]

From: Björn Lundin
<b.f.lundin@gmail.com>
Date: Thu, 3 Feb 2022 10:54:43 +0100

> a lot of effort in Ada programming goes into making programs never crash. For example, if the program runs out of memory during operation it might crash, so Ada programs are often written to never allocate new memory after a startup phase. In C++ or Rust, it's important not to get wrong answers like 2+2=5, but if your program runs out of memory and crashes, go get a bigger computer. So idiomatic C++ and Rust programming uses dynamic allocation freely, and that makes some kinds of programming more convenient, at the expense of tolerating possible crashes.

That depends on the domain. Perhaps it is true in embedded. I use Ada for large systems, talking to databases, and lots of administrative work. On a win/Unix server. We would not dream of pre-allocate memory at startup. We allocate and dispose as needed.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Thu, 3 Feb 2022 21:20:08 -0600

> How else would you do controlled types [if not with tagged types]?

Ada 9x originally had a bunch of magic attributes (similar to streaming). It was very complex and got dumped in the dustbin during "scope reduction". Later on, some of us were bemoaning that a critical feature (finalization) had gotten lost in Ada 9x, and Tucker came up with the idea to build it on top of tagged types as a massive simplification (at the loss of a bit of power). People often complain that Ada finalization is complex, and it is, except all of the alternatives are a lot more complex. :-)

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Thu, 3 Feb 2022 21:38:16 -0600

...

> Well are you familiar with red-black trees? They are a data structure similar to B-trees which you may have seen. Basically the trees have nodes that are coloured either red or black, and there are some rules such as that internal red nodes can have only black children, and that enforces some invariants that keep the trees balanced so that lookups and updates are guaranteed to run fairly fast.

> Now these trees have been implemented in many languages, and if you look at almost any implementation, there is usually a test suite or internal debugging code to make sure that the invariants are preserved after doing an update. It is quite easy to make a mistake after all. But the Haskell code doesn't have those tests. Why not? Because the invariants are enforced by the datatype! If you make a mistake and mess up an invariant, your code won't compile! It's the difference between checking a subscript at runtime, and verifying that it in range with SPARK. SPARK lets you get rid of the runtime check. Haskell's type system is able to do similar things.

Cool, and most likely useless. OOP is like that in many ways, it lets you make a bunch of static checks, but to get them, you have to contort your design in awful ways. And then it is hard to extend, as you have a huge number of routines to override to get anything done.

[...]

The key for a programming language design is to minimize what Ada calls erroneous execution (undefined behavior), because it is that possibility which stop proofs in their tracks (or at least should; at least some tools ignore that possibility and essentially give garbage results as a consequence). Ada needs work in that area, but most other languages need more -- Ada at least detects most problems with dynamic checks.

Anyway, that's my 20 cents (inflation, you know). :-)

From: Paul Rubin
<no.email@nospam.invalid>
Date: Thu, 03 Feb 2022 21:19:14 -0800

> In any language with dynamic checks, one can easily reduce the problem of correctness down to simply proving that no check fails.

Well sure, but I'd take issue with the word "easily" there. The invariants for the red-black tree are fairly complicated. Enforcing them could be done with what I think SPARK calls a "ghost function" or something like that: a piece of code that is not actually executed, but is used only by the prover. But, what would the proof look like I think it would likely involve some manual work with an external proof assistant like ISABELLE. That takes quite a bit of effort and knowledge on the programmer's part.

On the other hand, the RB tree type declaration in that Haskell example is very natural and appealing even in that old implementation, and using newer GHC features that have appeared since then, it becomes even nicer.

[...]

From: Luke A. Guest
<laguest@archeia.com>
Date: Fri, 4 Feb 2022 10:28:33 +0000

> Ada 9x originally had a bunch of magic attributes [for finalization]

Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

From: Andreas Zeurcher
<zuercher_andreas@outlook.com>
Date: Fri, 4 Feb 2022 09:51:59 -0800

> Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

Randy, I agree with Luke: were these intermediate design proposals lost entirely or have they (as still extant) have simply not been released publicly? I suspect that at least some of these attributes have nowadays analogues in C++ smart pointer's & Objective-C/Swift's ARC {strong, weak, plain old data not needing finalization, pointed-to-object ownership, presence of finalization

subroutine/function/procedure a.k.a. finalizer/destructor, whether this finalizer in a subtype displaces its parent's finalizer versus this finalizer in a subtype chains its finalizer to all of its ancestors' finalizers unwound from most-derived to underived-root, … and so forth}. Or was Tucker's set of magic attributes going an entirely different direction? That intermediate proposal under consideration back in the 1st half of the 1990s might be a quite interesting read (especially by a reader with an interest in envisioning an Ada-esque analogue of Rust's borrow-checker algorithm).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 4 Feb 2022 22:31:40 -0600

> Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

They were specifically for finalization, and got called automatically in various places.

Re: Andreas. So far as I know, the documents existed only on paper - there never were any electronically distributed versions outside of the Ada 9x team (and possibly the printers). I still have a set of them on my bookshelf -- I look at them periodically to see where current oddities appeared in the language (and possibly to get some idea why). [But see below.]

Looking in the RM 3.0 (the final version was 6.0 for reference), it already had the tagged type version, but they were derived from an implementation defined type "Finalization_Implementation", and what became Adjust was named Duplicate.

Looking in ILS 1.2 (a working document full of ideas but not quite a complete RM, dated Dec 1992), I can't find any sign of finalization. It must have been gone by then.

I do have a large number of older documents somewhere in storage, but this isn't worth digging around in there to find out. Most of those were incomplete design documents.

You might be able to find something about that design in the Ada 9x mail archive or in the LSNs (Language Study Notes). You can find them in the AdaIC archives. Rooting around in there, there are some promising looking documents in the "history" section of the AdaIC archives. There is a directory of stuff called "9x-history"; there probably is interesting stuff there.

http://archive.adaic.com/pol-hist/history/9x-history/

LSNs are found in:
http://archive.adaic.com/standards/95lsn/

The Ada 9x mail archive (These were known as "MRT comments"):

http://archive.adaic.com/standards/95com/mrtcomments/

The comments of interest here are probably in the ZIPed comments rather than the more recent ones. (These are text files, I think, even though they don't have a usual extension.)

From: amo...@unizar.es <amosteo@unizar.es>
Date: Fri, 11 Feb 2022 09:40:14 -0800

> If possible please tell what Rust has to offer over Ada.

In my minimally informed opinion after going through parts of the official tutorial a couple of times, what Rust has to offer in general:

+ Memory safety (no leaks, double-free, race conditions*) by default.

- Terrible illegible syntax.

+ Safer/more expressive/more modern constructs than C.

+ Modern tooling shared by all the community.

[*] I guess in a protected-object sense, not in a high-level logic sense. But I don't really know.

The thing is that C is so basic and C++ so prone to shooting yourself in the foot, that Rust hits a middle ground that feels like the best thing since sliced bread to C/C++ programmers wishing for something better. Add to that the true novel contribution to a mainstream language that is memory safety (this is really a new way of thinking when you get into Rust), that if you don't know better (e.g., Ada) it really is overwhelmingly compelling. I'm not surprised at the cult-like following (I mean, we feel like that sometimes in the Ada world, right?) In a sense, Rust is the Ada of a younger generation, and without the baggage.

Of course you sometimes have to use "unsafe" programming evading the borrow checker, just like pointers are sometimes a necessity in Ada; and the legibility becomes truly awful IMHO really fast (to me, this is THE Achilles heel nobody seems to care too much about), but as I said, it has a couple of real selling points over the competition. Of course, if legibility is not your concern because you're used to C++ templating nightmares, you don't feel that particular pain. It's always the same story with Ada; most people don't know better to realize what they're missing.

The whole memory safety thing with the borrow checker goes beyond a gimmick, and it has a solid quality which goes beyond "in Ada you don't need pointers most of the time". It's a compile-time check, and it makes evident that runtime checks are a poor substitute. I'm more ashamed now of the whole anonymous pointers and accessibility surprises in Ada. Yes, SPARK added something similar for pointers, but in Rust it is for all variables. The equivalence in Ada would be not being able to use the same variable in two consecutive calls as an in-out parameter. So it's not the same, besides being only in SPARK.

Not having done anything of real import, I'm not sure how inevitable it is to go unsafe in Rust. My guess is that it will be hidden in libraries just like the Ada standard containers contain some scary pointer use (and I mean that I wouldn't like to have to understand what is going on there with the tampering checks etc.) At that point, obviously, you've lost the most solid selling point IMHO. Ada is safer not in a single sense, but as a whole design.

All in all, Rust has one big thing that Ada hasn't, which is the borrow checker.

And that is how I would summarize it: Rust is better in a single narrow sense, but Ada is better as a general language. Which is, not surprisingly, the consequence of the design motivations for each, which were precisely to have a memory-safe language and a high-integrity-oriented language. So the funny thing is that both have succeeded at their objective.

I really miss not having the time to become proficient in Rust at least to be able to properly compare. I think the memory safety is great to have (and if Ada were designed today, I guess it should play the same integral part, if practical), but Rust is demanding on the programmer in a way that C/C++ aren't, and the maintainability seems suspect, so I don't know how far it will carry Rust into the future. I guess it could absorb a big part of both new C and C++ development.

Boy, can I write a lot sometimes...

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 11 Feb 2022 19:24:02 +0000

> The thing is that C is so basic and C++ so prone to shooting yourself in the foot, that Rust hits a middle ground that feels

I'd say C++ is like a "backgun" (search images).

> like the best thing since sliced bread to C/C++ programmers wishing for something better. [...]

Exactly, people want something better, but for some reason CAN NOT accept anything that doesn't look like C/C++.

> [...] It's always the same story with Ada; most people don't know better to realize what they're missing.

And refuse to look for better, just accepting to continue using the same old, same old.

> The whole memory safety thing with the borrow checker goes beyond a gimmick, and it has a solid quality which goes beyond "in Ada you don't need pointers most of the time". It's a compile-time check, and it makes evident that runtime checks are a poor substitute.

So, you'd prefer, if Ada was designed now, it didn't do runtime checks (on pointers) and have compile-time checks?

> Yes, SPARK added something similar for pointers [...]

They added memory tracking at gnatprove time, much like the borrow checker afaik, which is an additional step.

[...]

> All in all, Rust has one big thing that Ada hasn't, which is the borrow checker.

I've not learnt any rust yet and that is my conclusion from what I've read. I need to do some tutorials at some point, but I also need eye bleach.

[...]

From: John Perry <devotus@yahoo.com>
Date: Fri, 11 Feb 2022 21:22:50 -0800

> I really miss not having the time to become proficient in Rust at least to be able to properly compare.

I've followed this thread with some interest. I've had to learn & use Rust at work; it has its ups and downs.

> + Memory safety (no leaks, double-free, race conditions*) by default.

Here's what Rust promises: https://doc.rust-lang.org/nomicon/races.html

"Safe Rust guarantees an absence of data races, which are defined as... [omitted] Rust does not prevent general race conditions. This is pretty fundamentally impossible, and probably honestly undesirable. ... So it's perfectly "fine" for a Safe Rust program to get deadlocked or do something nonsensical with incorrect synchronization. ... Still, a race condition can't violate memory safety in a Rust program on its own."

> In a sense, Rust is the Ada of a younger generation, and without the baggage.

Not quite. It's kind of discouraging to me how many of the older generation roll their eyes when you mention Ada, or relate stories of how it didn't work out for previous places of employment.

> Of course you sometimes have to use "unsafe" programming evading the borrow checker, just like pointers are sometimes a necessity in Ada...

The only time I've found it necessary (so far) to use the "unsafe" keyword is when interfacing with C or C++. There are people, and probably entire fields of IT, where "unsafe" may be much more common.

> and the legibility becomes truly awful IMHO really fast (to me, this is THE Achilles heel nobody seems to care too much about

I agree with this. It's not as bad as C++, not even as bad as C IMHO, but the braces get old. If not for IDEs that can help navigate them, I'd get lost pretty easily.

> The whole memory safety thing with the borrow checker goes beyond a gimmick [...] It's a compile-time check

In addition, the compiler's error messages are *very* useful, better than GNAT's for certain. The only time we have trouble making sense of them is, again, when we have to interface with C or C++ code.

(Well, except when I was learning. I got pretty frustrated with the compiler error messages at times. And I still haven't figured out Rust's manner of organizing modules; if I do understand it, then "lib.rs" is a much bigger deal than I think it should be. But I probably just don't understand well enough yet.)

> [...] The equivalence in Ada would be not being able to use the same variable in two consecutive calls as an in-out parameter.

Maybe I misunderstand, but I think the analogy's incorrect. When you designate a Rust variable as mutable, you can in fact have two consecutive calls in a manner akin to "in out", _so long as_ the function declares it wants to "borrow" the variable as mutable, *and* so long as the caller gives permission for both the borrow and the mutability. If it doesn't, the compiler gives a very clear error message.

I'm not sure Ada has anything comparable to that.

> Not having done anything of real import, I'm not sure how inevitable it is to go unsafe in Rust.

At work we have a fairly non-trivial Rust system that, as far as I know, goes unsafe only when... you can fill in the blank. :-)

> And that is what how I would summarize it: Rust is better in a single narrow sense, but Ada is better as a general language.

I haven't played with Ada's task & rendezvous mechanisms in a long time. Do they guarantee an absence of data races? If not, I'd say that's something else Rust has that Ada doesn't. I think SPARK does guarantee that, though. (If I understand correctly, the key is to disallow mutable objects being passed to multiple tasks / threads / etc.)

> Rust is demanding on the programmer in a way that C/C++ aren't...

Perhaps, C/C++ are demanding on the programmer in all kinds of ways that Rust isn't, and none of those ways is good. ;-) Whereas Rust's demands are pretty much all good (in comparison to C/C++).

I would also add that Rust has an amazing and effective ecosystem of libraries that are extremely easy to download and build, all as part of the generally-used Cargo build tool, which as far as I can tell is much easier to use and much more robust than ant, gradle, make, etc. I have the impression that alire is inspired by Cargo, but I haven't used alire at all yet, so I don't know how it compares beyond the ability to create projects and download libraries. I also don't know if alire is nearly as comprehensive as what Cargo offers (see, for instance, https://crates.io/, which offers tens of thousands of crates, and https://docs.rs/, which documents them -- alire has about 220 crates).

I have a feeling that abundance of crates, and the ease of incorporating and using them, has at least as much appeal as the guarantees on any safe code you may write.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Sat, 12 Feb 2022 02:08:11 -0800

> > and the legibility becomes truly awful IMHO really fast [...]

> I agree with this. It's not as bad as C++ [...]

Agree too, but only because they use K&R style. I find Allman style quite readable.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Sat, 12 Feb 2022 18:34:04 +0100

> So, you'd prefer, if Ada was designed now, it didn't do runtime check (on pointers) and have compile-time checks?

I'd prefer that, as much as feasible, checks were moved (not removed!) to compile-time, yes. I know there are efforts in this direction at AdaCore to simplify the accessibility checks model.

>> I'm more ashamed now of the whole anonymous pointers and accessibility surprises in Ada.

> I'm not sure what you mean here.

My problem with runtime checks (which are undoubtedly better than no checks, sure), and in particular with accessibility checks, is that sometimes you get a failure much later during testing. By that time, understanding the problem may be 1) hard and 2) require painful redesign. At compile-time you get to deal with the problem immediately.

This is something in which Rust and Ada share the sentiment: "if it compiles, it works". So having something in another language found at compile-time makes me want to have it also in Ada at compile-time. It really spoils you against runtime checks. Much like I prefer the static elaboration model in GNAT instead of the dynamic one.

Also there are times in Ada where static checks are false positives that require some 'Unchecked_Access, and other times there is no failure yet you're doing something wrong. I find these from time to time in pretty obscure combinations not easy to provide a reproducer and frankly, I hate it. I'm never sure if I'm at fault, the compiler is at fault, or I've hit a corner case in the "heart of darkness". Nowadays I won't use a pointer even if it means obscene underperformance, until the thing is unavoidable.

There are also situations in which marking a parameter as aliased, even if you know it is already by reference (a limited/tagged type), will alter the things you can do with 'Access/'Unchecked_Access. There have been a couple of recent posts about that. Even if it's my fault, I find too hard to repeatably remember the finer details.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Sat, 12 Feb 2022 19:24:02 +0100

> I agree with this. It's not as bad as C++, not even as bad as C IMHO, but the braces get old.

For me, it's not so much the braces as the reference/lifetime &'!<> soup.

> Maybe I misunderstand, but I think the analogy's incorrect. [...]

Yes, it is as you say. It is not a perfect analogy, and rethinking a bit more about it it's possible I was wrong including non-pointers. In Ada there's no trouble by default either; you have to mark things aliased, or take an 'Access/'Unchecked_Access to start to get into trouble.

With tasking involved is another matter, there Ada provides no safety when using global variables.

> I'm not sure Ada has anything comparable to that.

No, I think that's the novelty in Rust, the single-ownership model.

> I haven't played with Ada's task & rendezvous mechanisms in a long time. Do they guarantee an absence of data races?

Not for global variables, yes for task-local ones. For any decent design you'd encapsulate any shared data in a protected object or task, which would give the same assurance as the bit you quoted for Rust. Then there's Pragma Detect_Blocking, but that will only work for protected operations, and two tasks getting in a mutual deadlock needs not to involve protected operations.

> If not, I'd say that's something else Rust has that Ada doesn't. I think SPARK does guarantee that, though. (If I understand correctly, the key is to disallow mutable objects being passed to multiple tasks / threads / etc.)

I agree here. Rust prevents misuse of global variables at the low level of simultaneous access (from what you referenced before). This certainly can be useful in refactorings going from a single- to a multi-threaded design. In Ada you'd have to inspect every package for global state. SPARK deals with that, of course, but again: not Ada.

>> Rust is demanding on the programmer in a way that C/C++ aren't...

>

> Perhaps, C/C++ are demanding on the programmer in all kinds of ways that Rust isn't, and none of those ways is good. ;-) Whereas Rust's demands are pretty much all good (in comparison to C/C++).

Sure, that's a good point: it's not easy but it's for a good cause. That's another point I see in common with the Ada compiler. Still, I feel Ada is simpler for beginners. You don't need to face the harshness of the more complex aspects of the language, perhaps owing to simpler imperative roots. In Rust you must face the borrow checker head on.

> I would also add that Rust has an amazing and effective ecosystem of libraries that are extremely easy to download and build, all as part of the generally-used Cargo build tool,

The ecosystem certainly is a selling point. Look at Python; for quick and dirty there's nothing better simply because you know there's going to be the library you need.

> I have the impression that alire is inspired by Cargo, but I haven't used alire at all yet, so I don't know how it compares beyond the ability to create projects and download libraries.

The inspiration is there in a broad sense, but Alire is much younger and the manpower behind it is a fraction. For now we strive to cover the basics. There's also ideas from opam, virtualenvs, ... In some aspects Alire may be even more comprehensive (like crate configuration).

> I also don't know if alire is nearly as comprehensive as what Cargo offers [...] alire has about 220 crates).

No need to guess, given the difference in size of the communities.

> I have a feeling that abundance of crates, and the ease of incorporating and using them, has at least as much appeal as the guarantees on any safe code you may write.

Sure it's appealing. In many (most?) cases, as you say, it can tip the scales. Still, I don't think anyone that keeps on using Ada is doing it for the amount of ready-to-use libraries. It's a problem to attract new people, though. And it may disqualify Ada when a particular dependency is important and too costly to bind.

From: John Perry <devotus@yahoo.com>
Date: Sat, 12 Feb 2022 15:59:33 -0800

First, I agree with Alejandro about the reference/lifetime soup. The other day I saw an expression along the lines of &[&something] and thought, "what"? If I look & think hard enough, I can figure out what that means, but words would be so much nicer, and searching for the meaning of a word is a bit easier than searching for "[]".

On the other hand, again: the tooling and error messages are really very good. "cargo clippy" gives a lot of helpful advice/lint. I think one of the regular correspondents here sells or maintains one for Ada, but I can't remember the name [AdaControl —arm].

Anyway, I want to walk back part of what I said about Rust's safety, related to a post in a reddit thread where someone writes, "having students develop in Ada lead to them jumping from exception to exception until it worked, while other students writing code for the same problem in Rust lead to them swearing for 4 days until their code compiled and then being surprised that their code works, 100% as they expected and not ever producing a runtime error until the end of the two week practicum."

https://www.reddit.com/r/ada/comments/7wzrqi/why_rust_was_the_best_thing_that_could_have/

Maybe, but long term I'm not so sure.

Rust doesn't have a null in safe mode, so the idiomatic way to indicate an error is via a Result enum, which has two variants: Ok(result), where "result" is the desired result, or Err(msg), where "msg" is an error message.

https://doc.rust-lang.org/std/result/enum.Result.html

[...] In any case, handling the Result can be a little tedious (only a little but still!) so people often use the standard library's ".unwrap()" function instead. That means something akin to, "I'm confident the preceding expression had an Ok result, so just hand me the result. If it's an Err then go ahead and panic with Err's msg."

[...] But a lot of Rust users default to .unwrap() all the same, which makes me think that issue about Ada users jumping

from exception to exception may be a feature of a lot of Rust code, too. Depends on the self-discipline, I guess.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 13 Feb 2022 09:10:01 +0100

> In Ada you'd have to inspect every package for global state.

AdaControl has a rule for that: Global_References

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Feb 2022 17:25:54 -0600

> Not for global variables, yes for task-local ones.

Ada 2022 has "conflict checking" to detect and reject bad uses of global variables. It's especially important for the parallel constructs (for which you don't have the syntactic guardrails that you get with tasks. It doesn't prevent every data race, but it eliminates most of them. (You can still get in trouble with accesses to multiple objects; that isn't necessarily safe even if accesses to the individual objects are.)

But, so far as I know, GNAT doesn't implement it yet.

From: Paul Rubin <no.email@nospam.invalid>
Date: Mon, 14 Feb 2022 20:29:34 -0800

> It doesn't prevent every data race, but it eliminates most of them

I wonder if we need software transactional memory: https://research.microsoft.com/en-us/um/people/simonpj/papers/stm/stm.pdf

https://research.microsoft.com/~simonpj/Papers/stm/beautiful.pdf

In the second paper, jump to the bottom of page 7 to skip a bunch of introductory stuff.

From: Kevin Chadwick <kevc3no4@gmail.com>
Date: Fri, 18 Feb 2022 05:24:15 -0800

> If possible please tell what Rust has to offer over Ada.

I haven't written a single line of Rust and do not intend to but I have done some research before and after choosing Ada, to confirm my choice due to Rusts popularity. My biggest preference is Ada's readability, of course some love brevity even when it adds complexity for some reason that I cannot understand.

Ada's type system has already been mentioned, but is fantastic.

Another is that Ada has a focus on stack and has tried to offer more heap tools in recent decades.

Rust has a focus on heap. I prefer the simpler stack default! Personally I avoided the heap, even with C.

I have heard that Rusts ownership model can cause problem with local/stack usage of owned memory (podcast interviewing a core maintainer "Rust with Steve Klabnik" but from 2015).

I have seen Rusts unsafe used even for simple things in embedded Rust whilst removing ALL of their 3 protections. Whereas with Ada you can more precisely permit pointer use and rarely need to.

https://docs.rs/svd2rust/0.19.0/svd2rust/#peripheral-api

```
struct PA0 { _0: () }
impl PA0 {
    fn is_high(&self) -> bool {
        // NOTE(unsafe) actually safe
        // because this is an atomic read with
       // no side effects
        unsafe { (*GPIOA::ptr()).idr.read().bits()
& 1 != 0 }
    }
    fn is_low(&self) -> bool {
        !self.is_high()
    }
}
```

Ada has been engineered to avoid pointer use, which I find appealing. Rust users would cite memory flexibility as appealing.

"Why Pascal is Not My Favorite Programming Language" by Kernighan is sometimes brought up, though much of it does not apply to Ada and no longer applies in any case and is clearly biased. Does he really promote the use of #include! Personally I blame flexibility points of view like his as the primary cause, as to why I have critical updates on my phone every single month and spend many days per year vulnerable to known exploits. Though really it is management at Vendors relentlessly pushing C. Maybe Rust can shift that closed point of view? I am aware that if my business does not succeed then the opportunity to write Ada, may go with it.

WRT compile time checks vs runtime: GO was written precisely because its authors were fed up waiting for C++ to compile. For me it is not important but worth bearing in mind. Personally I like the simplicity of runtime checks. I have much more faith in them than compile time checks! Though I confess to not knowing the details well enough to make that statement with complete confidence. It would also be nice to handle them more easily in a ZFP runtime.

SPARK sounds great and I like how it is intended to be applied where needed but I am dubious of any text that says it proves this or that, when it often depends on the tests implemented. I much prefer the language used by one AdaCore member in a podcast (Paul Butcher) along the lines of providing a high degree of confidence/assurance.

Ada versus Pascal

From: 711 Spooky Mart <711@spooky.mart>
Subject: Ada versus Pascal
Date: Thu, 21 Oct 2021 22:29:15 -0500
Newsgroups: comp.lang.ada

The little snippets of Ada code I've seen look _alot_ like Pascal.

What degree of learning curve is there to learn Ada, coming from a Pascal background? What kind of rough timeframes to get comfortable with programming without always looking into the manuals?

Where is the best starting point for a Pascal programmer to get up and running with Ada?

From: Ldries46 <bertus.dries@planet.nl>
Date: Fri, 22 Oct 2021 08:18:07 +0200

> Where is the best starting point for a Pascal programmer to get up and running with Ada?

I learned programming in 1966/1967 in Algol 60. As seen in the Algol report that can be found on the internet, Algol 60 is mostly a language for defining algorithms. It does not define Input and Output procedures. Pascal is one of the languages that have Algol 60 as a predecessor as is Ada. I did learn Pascal from some course and later on I did learn Ada, the latter by just reading the book "Software Engineering with Ada" by Grady Booch. That was Ada 85, the first version of Ada. Ada is stricter than other languages and is meant to have NO Operating system dependent items, so if you cannot go around something there must be a package on each operating system having the same interface everywhere.

From: Paul Rubin <no.email@nospam.invalid>
Date: Thu, 21 Oct 2021 23:40:51 -0700

> What degree of learning curve is there to learn Ada [...]

Ada is not that hard to get started with, but it has orders of magnitude more stuff in it than Pascal does, and getting familiar with all the intricacies is a big task. I've fooled with Ada a little, I consider myself a language geek, I've been following this newsgroup on and off for years, but I can't understand that many of the discussions I see here.

If I wanted to do something serious with Ada, I'd start by working through an in-depth textbook rather than just an intro or tutorial.

For just getting started, I used "Ada Distilled" (easy to find online). It is pretty good, but there is an enormous amount of material that it doesn't cover.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Fri, 22 Oct 2021 11:57:17 +0300

> The little snippets of Ada code I've seen look _alot_ like Pascal.

Yes. Pascal syntax had a lot of influence on Ada syntax. But, as others have said, (current) Ada has a lot more features than (original) Pascal.

Very roughly speaking, and off the cuff, Ada has evolved as follows, which also gives you a list of the main things to learn, in addition to the Pascal base:

- Ada 83: Pascal + much improved type system + modules (packages) + exception handling + generic programming + concurrency (tasks)
- Ada 95: added modular (unsigned) integer types, object-oriented programming (classes = tagged types), package hierarchies (child packages), and asynchronous inter-task communication (protected objects)
- Ada 2005: added Java-like interface types (limited multiple inheritance), a standard container library, and further standard libraries
- Ada 2012: added support for program proof and logical run-time checks (preconditions, postconditions, type predicates), more forms of expressions (if expressions, case expressions, quantified expressions), and more standard libraries.
- Ada 2022 (upcoming): adds fine-grained parallel execution, extends the ability to do static (compile-time) computations, adds library packages for arbitrary-precision and arbitrary-range integer and real arithmetic ("bignums"), and makes lots of sundry improvements.

> What degree of learning curve is there to learn Ada?

As you can see from the list above, there is quite a lot to learn before you know _all_ of Ada. A Pascal-like subset should not be hard to learn, and if you learn the rest of the features in more or less the same order as they were added to Ada, you will pass from one consistent, working language subset to a larger such subset at each step.

The only point where I suggest to learn features in a different order is in inter-task communication: asynchronous communication via protected objects is much easier than was the original, synchronous rendez-vous method in Ada 83 (but which is of course still supported).

> Where is the best starting point for a Pascal programmer to get up and running with Ada?

I think this depends a lot on how you like to learn - by reading technical text (manuals) or by experimentation. I'm a "read the manual" type (I learned Ada from the Ada 83 Reference Manual) so perhaps I would start with the Ada Wikibook at https://en.m.wikibooks.org/wiki/Ada_Programming, which extends up to the 2012 Ada standard (or so it claims, I haven't really read it, but I've heard good reports of it).

Perhaps you could take one of your smaller Pascal programs and translate it to Ada as a first step? As the next step, you could divide that program into packages, then add exception handling. And then take a new problem and write an Ada program from scratch.

Ask for help here, or in some other Ada forum, whenever in doubt about something.

From: 711 Spooky Mart
<711@spooky.mart>
Date: Fri, 22 Oct 2021 04:59:25 -0500

> Ada is stricter than other languages and is meant to have NO Operating system dependent items, so if you cannot go around something there must be a package on each operating system having the same interface everywhere.

By this do you mean the same syntax and libs will run on all target systems without fiddling with {IFDEF} and architecture compiler switch woo foo for USES and repetitive cross-arch boilerplate?

One thing I can't stand about Pascal is the totally different functions and logic from several operating systems that MUST be re-written several times in the same code base to do the same job. This drives me mad. In fact it irks me so much I was thinking of writing some libraries for things I do that would handle this all automatically across arches. There would go a couple months of Sundays.

Think IPC with Pascal. Get a good IPC routine going for Linux in your app, then you have to re-write it for MAC and Windows, and even some other flavors of *nix.

So am I to understand that the Ada compiler has somehow eliminated this problem, by ensuring every target OS has a syntactically conformant package to execute its methods using the same statements?

If I'm understanding you rightly, even though Ada sounds like a much more complex language than Pascal, it also sounds like it would have less surprises across arches.

Please elaborate if I'm misunderstanding. And thanks to everyone else who has responded.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 22 Oct 2021 13:49:11 +0200

> What degree of learning curve is there to learn Ada, coming from a Pascal background?

Pascal was the starting point for the Green language, which became Ada in 1980 (and also for the Blue, Red, and Yellow languages, which did not). Ada is firmly in the ALGOL family of languages.

There is a sequential subset of Ada that Pascal users can learn very quickly: the sequential language + packages (packages [modules] are fundamental to Ada, and you can't do anything useful without them). One should then quickly learn generics, as much of the standard library is generic. Ada's features are mostly orthogonal, so one can use this subset without surprises from the other aspects of the language.

One can then learn programming by type extension (tagged types and interfaces) and concurrent programming (tasks and their friends) to complete your understanding of the language.

I generally recommend “Ada Distilled”

(https://www.adaic.org/wp-content/uploads/2010/05/Ada-Distilled-24-January-2011-Ada-2005-Version.pdf)

to those familiar with another imperative language. It's ISO/IEC 8652:2007 Ada, but you can easily pick up the new Ada-12 features when you've finished. (There's also now an Ada-12 version available on Amazon.)

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Fri, 22 Oct 2021 18:12:49 +0300

> By this do you mean the same syntax and libs will run on all target systems without fiddling with {IFDEF} and architecture compiler switch woo foo for USES and repetitive cross-arch boilerplate?

I'm not ldries46, but here is an answer: Ada standardizes _some_ functions for which some other languages use "OS" services, principally threading, which in Ada is the "tasking" feature. Indeed Ada tasking works in the same way whichever OS is used, and also in the "bare board", no-OS situation. This is very useful for developing multi-threaded embedded SW, because the Ada tasking code can be tested on desk-top workstations and then executed on the target system unchanged. (and no "ifdefs").

But real operating systems (as opposed to simpler real-time kernels) provide many services that are not standardized in Ada, for example inter-process communication.

> Think IPC with Pascal. Get a good IPC routine going for Linux in your app, then you have to re-write it for MAC and Windows, and even some other flavors of *nix.

Indeed.

> So am I to understand that the Ada compiler has somehow eliminated this problem, by ensuring every target OS has a syntactically conformant package to execute its methods using the same statements?

Sadly no.

However, there are some rudiments:

- There is a standardized Ada interface (binding) to POSIX services. This is implemented in an Ada library called Florist. If you find or make a Florist implementation for the OSes you use, your Ada program can use the same OS service interfaces on all those OSes.
- The gcc-based Ada compiler GNAT comes with a GNAT-specific library that provides some OS services with the same Ada API on any OS that GNAT supports. This includes some IPC, but I don't know exactly how far that goes, and the library may of course change from one GNAT version to the next.
- There is an Ada library called Win32Ada that provides an extensive set of Microsoft Windows services, but it is a "thin binding" meaning that even the API is Windows-specific.

The Ada applications I have created and worked with have needed only a few OS services, basically some IPC: text in and out via pipes to and from a child process. We implemented our own binding to the required OS services (pipe and process creation and destruction). The interface consisted of a package declaration (.ads file) that was basically the same for all the supported OSes (Windows, Linux, Mac OS X) but had different OS-specific package body files (implementations, .adb files). In practice I think the Linux and Mac OS X implementations were the same and used direct binding to fork() and pipe() etc. The Windows implementation used Win32Ada.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Fri, 22 Oct 2021 17:47:59 +0200

> But real operating systems (as opposed to simpler real-time kernels) provide many services that are not standardized in Ada, for example inter-process communication.

Ada 83 predates threads. Initially a task meant to be either scheduled internally or mapped onto system processes. It is not late now. One could allow the pragma Import for tasks (and protected objects) in order to communicate to an external process using rendezvous and protected actions.

> - The gcc-based Ada compiler GNAT comes with a GNAT-specific library that provides some OS services with the same Ada API on any OS that GNAT supports.

It provides sockets and serial I/O, one or both are vital for many applications.

- There is the annex E providing RPC and shared objects. Unfortunately it is very vague and underdocumented. [...]
- The simple components library provides inter-process communication primitives: mutexes, events, streams, pools, RPCs etc.

From: Dennis Lee Bieber
<wlfraed@ix.netcom.com>
Date: Fri, 22 Oct 2021 13:05:01 -0400

>The little snippets of Ada code I've seen look _alot_ like Pascal.

No surprise. The teams that took part in the DoD competition to design a language to replace the mish-mash of languages being used in the 70s tended to choose Pascal as the starting point (Modula-2 hadn't escaped ETH-Zurich yet <G>).

The main difference is that Ada incorporated block closing syntax at the base, finding Pascal (and C) [begin/end, {/} respectively] usage error-prone (dangling else, etc.) along with using ; as a terminator instead of separator. Oh, and using (/) for both function arguments and array indexing (back then, most US keypunches didn't support [/] or {/}).

Declarations do not have a defined sequence (type, constant, variable).

Also, Pascal of the era typically did not support separate compilation and/or include files -- programs were all single monolithic files, any change required recompiling the entire program.

Pascal also had a relatively limited I/O system -- with the bad quirk that it did "pre-reads" of files. Made interactive/console programs difficult (or required special handling by the run-time startup) -- starting a program would result in stdin reading at least one character, if not one line, into the file buffer variable... But the program may not want the data until after lots of initialization and prompts.

>What degree of learning curve is there to learn Ada, coming from a Pascal background?

If all one is writing is "Pascal" type applications, without using complex data types (ie; defining specific types for each "concept") -- it shouldn't take too long.

Tasking, rendezvous, protected objects (not to be confused with private objects), and generics, may take longer to get comfortable with.

The appendices of the LRM will tend to get lots of usage; there are many subtleties to the standard libraries.

From: Gautier Write-Only Address
<gautier_niouzes@hotmail.com>
Date: Fri, 22 Oct 2021 13:00:45 -0700

> What degree of learning curve is there to learn Ada, coming from a Pascal background?

You don't need to read more manuals than you were used to for Pascal (so, it can be a wide range, depending on your way of learning). The most outstanding difference between both languages is the degree of unification.

Here are a couple of links, with a Pascal-then-Ada perspective, that could be useful:

http://p2ada.sourceforge.net/pascada.htm

http://p2ada.sourceforge.net/

From: Paul Rubin
<no.email@nospam.invalid>
Date: Fri, 22 Oct 2021 17:29:26 -0700

> Also, Pascal of the era typically did not support separate compilation and/or include files

I thought Ada was originally like that too. The program could be split into multiple files, but they were expected to all be compiled together.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 22 Oct 2021 20:17:02 -0500

> I thought Ada was originally like that too.

No. Some implementations were like that, but most supported fully separate compilations from the beginning. Janus/Ada certainly did (once we got packages implemented, and Ada without packages really isn't Ada at all). You might have been thinking about the original permission to require generic bodies to be available when compiling an instantiation, but that only applied to generic units, never "regular" units. And some compilers (like Janus/Ada) never used that permission.

From: Dennis Lee Bieber
<wlfraed@ix.netcom.com>
Date: Sat, 23 Oct 2021 13:24:19 -0400

>I thought Ada was originally like that too. The program could be split into multiple files, but they were expected to all be compiled together.

No... Pretty much every build system for Ada focused on only rebuilding the parts affected by a changed file -- by following WITH statements to find required units (see the LRM for what a "unit" comprises) /and/ determining if that unit requires compilation. Timestamps or intermediate files may be used in that determination. Changes in implementation (body) require the body to be recompiled, but if the specification did not change, then units WITHing the specification don't need to be compiled -- they just need relinking with the updated body.

GNAT's build system -- using the host OS filesystem as the "database" -- required that separate files are generated for each unit. (cf: GNATCHOP) All-in-One was the optional source file format accepted by some compilers -- but other than the early language reference manuals, I haven't encountered any text books that use that means of presenting code examples (unless it is discussing the use of GNATCHOP itself <G>).

https://en.wikisource.org/wiki/Stoneman_requirements [Link replaced. See below Rosen's comment. —arm] is the requirements document that DoD used to define the desired environment around Ada development.

"""

4.E APSE TOOLSET REQUIREMENTS

4.E.1 The tools in an APSE shall support the development of programs in the Ada language as defined by the Ada reference manual. In particular an APSE shall support the separate compilation features of the language.

"""

Note the last sentence

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 24 Oct 2021 09:04:29 +0200

> https://www.adahome.com/History/Stoneman/stoneman.htm is the requirements document that DoD used to define the desired environment around Ada development.

Please don't provide links to adahome, this site is frozen since 1998, and there are copyright issues with the owner.

Stoneman can be obtained from:

https://en.wikisource.org/wiki/Stoneman_requirements

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 24 Oct 2021 09:04:29 +0200

> https://www.adahome.com/History/Stoneman/stoneman.htm is the requirements document that DoD used to define the desired environment around Ada development.

Please don't provide links to adahome, this site is frozen since 1998, and there are copyright issues with the owner.

Stoneman can be obtained from:

https://en.wikisource.org/wiki/Stoneman_requirements

From: Jerry <list_email@icloud.com>
Date: Sat, 23 Oct 2021 21:33:14 -0700

This thread began as a comparison of Ada and the original Pascal. So how does Ada compare to Free Pascal Compiler and Delphi which have gone far past original Pascal?

From: Ldries46 <bertus.dries@planet.nl>
Date: Sun, 24 Oct 2021 08:32:01 +0200

> how does Ada compare to Free Pascal Compiler and Delphi which have gone far past original Pascal?

You can also ask can you compile a Free Pascal program in Delphi or in the other direction. Ada was intended to do so and to keep it that way

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sun, 24 Oct 2021 09:51:26 -0700

> how does Ada compare to Free Pascal Compiler and Delphi [...]?

You find a very partial answer in the comparison here:

http://p2ada.sourceforge.net/pascada.htm#tables

Made around year 2000, so ~30 after original Pascal but ~20 years ago.

Note that both FPC and Delphi descend from Turbo Pascal, which is itself completely different from other extensions like ISO Extended Pascal.

In a nutshell, Pascal is an extreme example of fragmentation of a language into dialects.

Ada is on the other extremity: you can build the same source sets (I mean exactly the same sources, without preprocessing gimmicks) on completely different compilers & OSes.

From: 711 Spooky Mart <711@spooky.mart>
Date: Sun, 24 Oct 2021 18:24:21 -0500

> Ada is on the other extremity: you can build the same source sets (I mean exactly the same sources, without preprocessing gimmicks) on completely different compilers & OSes.

I think that answers the important original query.

Does modern Ada have facility for writing boot loaders, inline Assembly, kernels, etc.?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 25 Oct 2021 11:23:49 +0300

> Does modern Ada have facility for writing boot loaders, inline Assembly, kernels, etc.?

In-line assembly is supported by most of the Ada compilers I have used, but the syntax may differ across compilers.

The run-time systems (real-time kernels) associated with Ada compilers for bare-board embedded systems are typically written in Ada, with minor amounts of assembly language inserted for the very HW-specific parts such as HW context saving and restoring.

I'm not very familiar with boot loaders, but I see no reason why a boot loader could not be written in Ada. However, usually (and as for other languages) there will be a small start-up routine in assembly language to initialize the processor, set up a stack, and so forth. The "Ada Bare Bones" project is doing something like this, I believe:

https://wiki.osdev.org/Ada_Bare_bones

From: Luke A. Guest <laguest@archeia.com>
Date: Mon, 25 Oct 2021 09:40:38 +0100

Boot loaders and kernels are just another application area any general purpose language can target, even Ada.

> I'm not very familiar with boot loaders

If you're talking x86 on PC's, then you'll need to read up on the x86 boot process in which x86 starts up in 16-bit (real) mode, then has to be taken into protected and then long modes. You would need a GCC that can target all those modes.

> The "Ada Bare Bones" project is doing something like this, I believe

Thanks for pointing out my project :) It's out of date and doesn't build as is any more, but others have written Ada pages on that site since.

IIRC, the changes that need to happen is that gprbuild

--target=arm-<whatever> be used instead of arm-<whatever>-gprbuild

Ada Practice

Custom Storage Pool Questions

From: Jere <jhb.chat@gmail.com>
Subject: Custom Storage Pool questions
Date: Sun, 12 Sep 2021 17:53:47 -0700
Newsgroups: comp.lang.ada

I was learning about making user defined storage pools when I came across an article that made me pause and wonder how portable storage pools actually can be. In particular, I assumed that the Size_In_Storage_Elements parameter in the Allocate operation actually indicated the total number of storage elements needed.

```
procedure Allocate(
    Pool : in out Root_Storage_Pool;
    Storage_Address : out Address;
    Size_In_Storage_Elements : in
    Storage_Elements.Storage_Count;
    Alignment : in
    Storage_Elements.Storage_Count)
is abstract;
```

But after reading the following AdaCore article, my assumption is now called into question: https://blog.adacore.com/header-storage-pools

In particular, the blog there advocates for separately counting for things like unconstrained array First/Last indices or the Prev/Next pointers used for Controlled objects. Normally I would have assumed that the Size_In_Storage_Elements parameter in Allocate would account for that, but the blog clearly shows that it doesn't

So that seems to mean to make a storage pool, I have to make it compiler specific or else risk someone creating a type like an array and my allocation size and address values will be off.

Is it intended not to be able to do portable Storage Pools or am I missing some Ada functionality that helps me out here. I scanned through the list of attributes but none seem to give any info about where the object's returned address is relative to the top of the memory actually allocated for the object. I saw the attribute Max_Size_In_Storage_Elements, but it doesn't seem to guarantee to include things like the array indices and it still doesn't solve the issue of knowing where the returned address needs to be relative to the top of allocated memory.

I can easily use a generic to ensure that the types I care about are portably made by the pool, but I can't prevent someone from using my pool to create other objects that I hadn't accounted for. Unless there is a way to restrict a pool from allocating objects of other types?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 13 Sep 2021 00:29:35 -0500

Not sure what you are expecting. There is no requirement that objects are allocated contiguously. Indeed, Janus/Ada will call Allocate as many times as needed for each object; for instance, unconstrained arrays are in two parts (descriptor and data area).

The only thing that you can assume in a portable library is that you get called the same number of times and sizes/alignment for Allocate and Deallocate; there's no assumptions about size or alignment that you can make.

If you want to build a pool around some specific allocated size, then if it needs to be portable, (A) you have to calculate the allocated size, and (B) you have to have a mechanism for what to do if some other size is requested. (Allocate a whole block for smaller sizes, fall back to built-in heap for too large is what I usually do).

More likely, you'll build a pool for a particular implementation. Pools are very low level by their nature, and useful ones are even more so (because they are using target facilities to allocate memory, or need to assume something about the allocations, or because they are doing icky things like address math, or ...).

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 13 Sep 2021 13:12:39 +0200

> I was learning about making user defined storage pools when I came across an article that made me pause and wonder how portable storage pools actually can be. [...]

That blog shows a special use for Storage_Pools, where you allocate /user/ data on top of the requested memory. When called by the compiler, it is up to the compiler to compute how much memory is needed, and your duty is to just allocate that.

From: Jere <jhb.chat@gmail.com>
Date: Mon, 13 Sep 2021 17:48:15 -0700

> That blog shows a special use for Storage_Pools

Yes, but if you look at that blog, they are allocating space for the /user/ data and for the Next/Prev for controlled types and First/Last for unconstrained arrays in addition to the size specified by allocate.

I agree I feel it is up to the compiler to provide the correct size to Allocate, but the blog would indicate that GNAT does not (or did not..old blog..so who knows?). Does the RM require that an implementation pass the full amount of memory needed to Allocate when new is called

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 14 Sep 2021 08:08:48 +0200

The RM says that an allocator allocates storage from the storage pool. You could argue that it does not say "allocates all needed storage...", but that would be a bit far fetched.

Anyway, a blog is not the proper place to get information from for that kind of issue. Look at the GNAT documentation.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 14 Sep 2021 08:23:08 +0200

> Yes, but if you look at that blog, they are allocating space for the /user/ data and for the Next/Prev for controlled types and First/Last for unconstrained arrays in addition to the size specified by allocate.

I do not understand your concern. The blog discusses how to add service data to the objects allocated in the pool.

I use such pools extensively in Simple Components. E.g. linked lists are implemented this way. The list links are allocated in front of list elements which can be of any type, unconstrained arrays included.

The problem with unconstrained arrays is not that the bounds are not allocated, they are, but the semantics of X'Address when applied to arrays.

A'Address is the address of the first array element, not of the array object. For a pool designer it constitutes a problem of getting the array object by address. This is what Emmanuel discusses in the blog.

[The motivation behind Ada choice was probably to keep the semantics implementation-independent.]

Consider for example a list of String elements. When Allocate is called with String, it returns the address of all String. But that is not the address you would get if you applied 'Address. You have to add/subtract some offset in order to get one from another.

In Simple Components this offset is determined at run-time for each generic instance.

Of course, a proper solution would be fixing Ada by adding another address attribute:

X'Object_Address

returning the first address of the object as allocated.

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 14 Sep 2021 08:42:52 +0200

> X'Object_Address

> returning the first address of the object as allocated.

But you cannot assume that the object is allocated as one big chunk. Bounds can be allocated at a different place. What would be X'Object_Address in that case?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 14 Sep 2021 09:00:13 +0200

The object address, without bounds, same as X'Address.

What Allocate returns is not what A'Address tells. The compiler always knows the difference, the programmer has to know it too. Nothing more.

From: Jere <jhb.chat@gmail.com>
Date: Tue, 14 Sep 2021 17:21:04 -0700

> I use such pools extensively in Simple Components. E.g. linked lists are implemented this way. The list links are allocated in front of list elements which can be of any type, unconstrained arrays included.

The blog I saw was old, so it is completely possible it no longer is true that GNAT does what the blog suggests. I'll take a look at your storage pools and see how they handle things like this.

From: Jere <jhb.chat@gmail.com>
Date: Tue, 14 Sep 2021 17:39:43 -0700

> Anyway, a blog is not the proper place to get information from for that kind of issue. Look at the GNAT documentation.

I'll take a look at the GNAT docs to see (and of course that blog is old, so GNAT may not do this anymore anyways), but I am mainly asking in the frame of what Ada allows and/or expects. I'd like to be able to allocate storage simply without worrying how the compiler does it under the hood and just assume that any calls to Allocate will ask for the full amount of memory.

Am I correct to assume that Ada doesn't provide any language means to restrict what types a pool can make objects of. The times that I have wanted to make a pool are generally for specific types and it is often simpler to design them if I can assume only those types are being generated

Thanks for the response. I'm sorry for all the questions. That's how I learn and I realize it isn't a popular way to learn in the community, but I have always learned very differently than most.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 15 Sep 2021 08:54:07 +0200

It seems that you are under the impression that Allocate must allocate more size than its Size parameter asks. The answer is no, unless *you* wanted to add something to each allocated object.

[...]

Again, if attributes are added, then it should be the object address as allocated. The compiler always knows the proper address because this address is passed to Free, not X'Address!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 15 Sep 2021 21:07:22 +0200

[Continuing on the subject of how to retrieve the object in the pool from its ‘Address, when there is hidden data like a fat pointer, Dmitry challenges readers to implement a function that needs to do so. —arm]

Now define and implement the following function:

```
type Type_Pointer is access Some_Type
     with Storage_Pool => Pool;
  function Get_Allocation_Time
       (Pointer : Type_Pointer) return Time;
```

The function returns the time when the pointed object was allocated in the pool.

[...]

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 16 Sep 2021 00:12:58 -0700

I am the original implementer of GNATCOLL.Storage_Pools.Headers, and I have been silent in this discussion because I must admit I forgot a lot of the details... To be sure, we did not add new attributes just for the sake of GNATCOLL, those existed previously so likely had already found other uses.

As has been mentioned several times in the discussion, the compiler is already passing the full size it needs to Allocate, and the storage pool only needs to allocate that exact amount in general. This applies for the usual kinds of storage pools, which would for instance preallocate a pool for objects of fixed sizes, or add stronger alignment requirements.

In the case of the GNATCOLL headers pool, we need to allocate more because the user wants to store extra data. For that data, we are left on our own to find the number of bytes we need, which is part of the computation we do: we of course need the number of bytes for the header's object_size, but also perhaps some extra bytes that are not returned by that object_size in particular for controlled types and arrays.

Note again that those additional bytes are for the header type, not for the type the user is allocating (for which, again, the compiler already passes the number of bytes it needs).

The next difficulty is then to convert from the object's 'Address back to your extra header data. This is when you need to know the size of the prefix added by the compiler (array bounds, tag,...) to skip them and then take into account the alignment, and finally the size of your header.

Dmitry's suggested exercise (storing the timestamp of the allocation) seems like a useful one indeed. It would be nice indeed if GNATCOLL's code wasn’t too complicated, but I am afraid this isn't the case. We had used those pools to implement reference counting for various complex types, and ended up with that complexity.

AdaCore (Olivier Hainque) has made a change to the implementation since the blog was published (https://github.com/AdaCore/gnatcoll-core/commits/master/src/gnatcoll-storage_pools-headers.adb), so I got some details wrong in the initial implementation apparently.

From: Jere <jhb.chat@gmail.com>
Date: Thu, 16 Sep 2021 16:21:58 -0700

> In the case of the GNATCOLL headers pool, we need to allocate more because the user wants to store extra data. For that data, we are left on our own to find the number of bytes we need, which is part of the computation we do: we of course need the number of bytes for the header's object_size, but also perhaps some extra bytes that are not returned by that object_size in particular for controlled types and arrays.

> Note again that those additional bytes are for the header type, not for the type the user is allocating (for which, again, the compiler already passes the number of bytes it needs).

Thanks for the response Emmanuel. That clears it up for me. I think the confusion for me came from the terminology used then. In the blog, that extra space for First/Last and Prev/Next was mentioned as if it were for the element, which I mistook was the user's object being allocated and not the header portion. I didn't catch that as the generic formal's name, so that is my mistake. I guess in my head, I would have expected the formal name to be Header_Type or similar so I misread it in my haste.

I appreciate the clarity and apologize if I caused too much of a stir. I was asking the question because I didn't understand, so I hope you don't think too poorly of me for it, despite my mistake.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 17 Sep 2021 15:56:20 +0200

> I was asking the question because I didn't understand, so I hope you don't think too poorly of me for it, despite my mistake.

Nope, especially because the issue with X'Address being unusable for memory pool developers is a long standing painful problem that needs to be resolved. That will never happen until a measurable group of people start asking questions. So you are doubly welcome.

From: Simon Wright <simon@pushface.org>
Date: Fri, 17 Sep 2021 20:46:26 +0100

> the issue with X'Address being unusable for memory pool developers is a long standing painful problem that needs to be resolved.

There are two attributes that we should all have known about, Descriptor_Size[1] (bits, introduced in 2011) and Finalization_Size[2] (storage units, I think, introduced in 2017)

[1] https://docs.adacore.com/live/wave/gnat_rm/html/gnat_rm/gnat_rm/implementation_defined_attributes.html#attribute-descriptor-size

[2] https://docs.adacore.com/live/wave/gnat_rm/html/gnat_rm/gnat_rm/implementation_defined_attributes.html#attribute-finalization-size

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 17 Sep 2021 22:39:05 +0200

They are non-standard and have murky semantics I doubt anybody really cares about. What is needed is the address passed to Deallocate should the object be freed = the address returned by Allocate. Is that too much to ask?

BTW, finalization lists (#2) should have been removed from the language long ago. They have absolutely no use, except maybe for debugging, and introduce huge overhead. The semantics should have been either Unchecked_Deallocation or compiler allocated objects/components may call Finalize, nothing else.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 00:17:50 +0300

> What is needed is the address passed to Deallocate should the object be freed = the address returned by Allocate. Is that too much to ask?

That is already required by RM 13.11(21.7/3): "The value of the Storage_Address parameter for a call to Deallocate is the value returned in the Storage_Address parameter of the corresponding successful call to Allocate."

The "size" parameters are also required to be the same in the calls to Deallocate and to Allocate.

> BTW, finalization lists (#2) should have been removed from the language long ago.

Huh? Where does the RM _require_ finalization lists? I see them mentioned here and there as a _possible_ implementation technique, and an alternative "PC-map" technique is described in RM 7.6.1 (24.r .. 24.t).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 09:49:16 +0200

> That is already required by RM 13.11(21.7/3): "The value of the Storage_Address parameter for a call to Deallocate is the value returned in the Storage_Address parameter of the corresponding successful call to Allocate."

You missed the discussion totally. It is about the X'Address attribute.

The challenge: write pool with a function returning object allocation time by its pool-specific access type.

> Huh? Where does the RM _require_ finalization lists?

7.6.1 (11 1/3)

> I see them mentioned here and there as a _possible_ implementation technique, and an alternative "PC-map" technique is described in RM 7.6.1 (24.r .. 24.t).

I don't care about techniques to implement meaningless stuff. It should be out, at least there must be a representation aspect for turning this mess off.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 12:03:03 +0300

> You missed the discussion totally. It is about X'Address attribute.

Sure, I understand that the address returned by Allocate, and passed to Deallocate, for an object X, is not always X'Address, and that you would like some means to get the Allocate/Deallocate address from (an access to) X. But what you stated as not "too much to ask" is specifically required in the RM paragraph I quoted. Perhaps you meant to state something else, about X'Address or some other attribute, but that was not what you wrote.

Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided. You would need some way of iterating over all the pieces allocated for a given object, or some way of defining a "main" or "prime" piece and a means to get the Allocate/Deallocate address for that piece.

>> Huh? Where does the RM _require_ finalization lists?

> 7.6.1 (11 1/3)

RM (2012) 7.6.1 (11.1/3) says only that objects must be finalized in reverse order of their creation. There is no mention of "list".

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 12:22:45 +0200

> Perhaps you meant to state something else, about X'Address or some other attribute, but that was not what you wrote.

I wrote about attributes, specifically GNAT-specific ones used in the blog to calculate the correct address. "Too much to ask" was about an attribute that would return the object address directly.

> Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided.

Such implementations would automatically disqualify the compiler. Compiler-generated piecewise allocation is OK for the stack, not for user storage pools.

And anyway, all this equally applies to X'Address.

> RM (2012) 7.6.1 (11.1/3) says only that objects must be finalized in reverse order of their creation. There is no mention of "list".

It talks about "collection."

> Then your complaint seems to be about something specified for the order of finalization, but you haven't said clearly what that something is.

No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 18:59:27 +0300

>> Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided.

> Such implementations would automatically disqualify the compiler.

That is your opinion (or need), to which you are entitled, of course, but it is not an RM requirement on compilers -- see Randy's posts about what Janus/Ada does.

[...]

> No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

So you want a way to specify that for a given access type, although the accessed object type has a Finalize operation or needs finalization, the objects left over in the (at least conceptually) associated collection should _not_ be finalized when the scope of the access type is left? Have you proposed this to the ARG?

To me it seems a risky thing to do, subverting the normal semantics of initialization and finalization. Perhaps it could be motivated for library-level collections in programs that are known to never terminate (that is, to not need any clean-up when they do stop or are stopped).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 18:19:23 +0200

> So you want a way to specify that for a given access type, although the accessed object type has a Finalize operation or needs finalization, the objects left over in the (at least conceptually) associated collection should _not_ be finalized when the scope of the access type is left?

Exactly, especially because these objects are not deallocated, as you say they are left over. If they wanted GC they should do that. If they do not, then they should keep their hands off the objects maintained by the programmer.

> To me it seems a risky thing to do, subverting the normal semantics of initialization and finalization.

Quite the opposite, it is the collection rule that subverts semantics because objects are not freed, yet mangled.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sun, 19 Sep 2021 13:36:11 +0300

> Quite the opposite, it is the collection rule that subverts semantics because objects are not freed, yet mangled.

Local variables declared in a subprogram are also not explicitly freed (deallocated), yet they are automatically finalized when the subprogram returns.

My understanding of Ada semantic principles is that any object that is initialized should also be finalized. Since the objects left in a collection associated with a local access type become inaccessible when the scope of the access type is left, finalizing them automatically, even without an explicit free, makes sense to me. If you disagree, suggest a change to the ARG and see if you can find supporters.

Has this feature of Ada caused you real problems in real applications, or is it only a point of principle for you?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 19 Sep 2021 13:41:00 +0200

> Local variables declared in a subprogram are also not explicitly freed (deallocated), yet they are automatically finalized when the subprogram returns.

Local objects are certainly freed. Explicit or not, aggregated or not, is irrelevant.

> My understanding of Ada semantic principles is that any object that is initialized should also be finalized.

IFF deallocated.

An application that runs continuously will never deallocate, HENCE finalize certain objects.

> Has this feature of Ada caused you real problems in real applications, or is it only a point of principle for you?

1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. I fail to see where that sort of thing might be even marginally useful. Specialized pools, e.g. mark-and-release will deploy their own bookkeeping, not rely on this.

2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

From: Jere <jhb.chat@gmail.com>
Date: Sun, 19 Sep 2021 17:31:26 -0700

> Not sure what you are expecting. There is no requirement that objects are allocated contiguously. Indeed, Janus/Ada will call Allocate as many times as needed for each object; for instance, unconstrained arrays are in two parts (descriptor and data area).

Followup question cause Randy's statement got me thinking: If a compiler is allowed to break up an allocation into multiple calls to Allocate (and of course Deallocate), how does one go about enforcing that the user's header is only created once In the example Randy gave (unconstrained arrays), in Janus there is an allocation for the descriptor and a separate allocation for the data. If I am making a storage pool that is intending to create a hidden header for my objects, this means in Janus Ada (and potentially other compilers) I would instead create two headers, one for the descriptor and one for the data, when I might intend to have one header for the entire object.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 09:34:47 +0300

I think one cannot enforce that, because the calls to Allocate do not indicate (with parameters) which set of calls concern the same object allocation.

This is probably why Dmitry said that such compiler behaviour would "disqualify the compiler" for his uses.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 19 Sep 2021 23:48:20 -0700

> I think one cannot enforce that, because the calls to Allocate do not indicate (with parameters) which set of calls concern the same object allocation.

I think the only solution would be for this compiler to have another attribute similar to 'Storage_Pool, but that would define the pool for the descriptor:

```
for X'Storage_Pool use Pool;
for X'Descriptor_Storage_Pool use Other_Pool;
```

That way the user can decide when to add (or not) extra headers.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 10:05:14 +0300

> Local objects are certainly freed. Explicit or not, aggregated or not, is irrelevant.

Objects left over in a local collection may certainly be freed automatically, if the implementation has created a local pool for them. See ARM 13.11 (2.a): "Alternatively, [the implementation] might choose to create a new pool at each accessibility level, which might mean that storage is reclaimed for an access type when leaving the appropriate scope."

> An application that runs continuously will never deallocate, HENCE finalize certain objects.

And I agreed that in this case it could be nice to let the programmer specify that keeping collections is not needed.

> 1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. [...]

Have you actually measured or observed that overhead in some application?

> 2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

So there is a possibility of programmer mistake, leading to unintended finalization of those (now inaccessible) objects.

However, your semantic argument (as opposed to the overhead argument) seems to be based on an assumption that the objects "left over" in a local collection, and which thus are inaccessible, will still, somehow, participate in the later execution of the program, which is why you say that finalizing those objects would "corrupt" them.

It seems to me that such continued participation is possible only if the objects contain tasks or are accessed through some kind of unchecked programming. Do you agree?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 20 Sep 2021 09:35:35 +0200

> Objects left over in a local collection may certainly be freed automatically, if the implementation has created a local pool for them. See ARM 13.11 (2.a): "Alternatively, [the implementation] might choose to create a new pool at each accessibility level, which might mean that storage is reclaimed for an access type when leaving the appropriate scope."

May or may not. The feature which correctness depends on scopes and lots of further assumptions has no place in a language like Ada.

> Have you actually measured or observed that overhead in some application?

How?

However you could estimate it from the most likely implementation as a doubly-linked list. You add new element for each allocation and remove one for each deallocation. The elements are allocated in the same pool or in some other pool. Finalization is not time bounded, BTW. Nice?

> It seems to me that such continued participation is possible only if the objects contain tasks or are accessed through some kind of unchecked programming. Do you agree?

No. You can have them accessible over other access types with wider scopes:

```
Collection_Pointer := new X;
Global_Pointer :=
Collection_Pointer.all'Unchecked_Access;
```

access discriminants etc. As I said, hands off!

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 11:08:52 +0300

> Global_Pointer := Collection_Pointer.all'Unchecked_Access;

So, unchecked programming, as I said.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 20 Sep 2021 10:28:28 +0200

> So, unchecked programming, as I said.

Right, working with pools is all that thing. Maybe "new" should be named "unchecked_new" (:-))

Finalize and Initialize certainly should have been Unchecked_Finalize and Unchecked_Initialize as they are not enforced. You can override the parent's Initialize and never call it. It is a plain primitive operation anybody can call any time any place. You can even call it before the object is fully initialized!

So, why bother with objects the user manually allocates (and forgets to free)?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:48:02 -0500

> The problem with unconstrained arrays is not that the bounds are not allocated, they are, but the semantics of X'Address when applied to arrays.

> A'Address is the address of the first array element, not of the array object. For a pool designer it constitutes a problem of getting the array object by address. This is what Emmanuel discusses in the blog.

Right, this is why Janus/Ada never "fixed" 'Address to follow the Ada requirement. (Our Ada 83 compiler treats the "object" as whatever the top-level piece is, for an unconstrained array, that's the bounds -- the data lies elsewhere and is separately allocated -- something that follows from slice semantics.)

I suppose your suggestion of implementing yet-another-attribute is probably the right way to go, and then finding every use of 'Address in existing RRS Janus/Ada code and changing it to use the new attribute that works "right".

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:51:15 -0500

Sorry about that, I didn't understand what you were asking. And I get defensive about people who think that a pool should get some specific Size (and only that size), so I leapt to a conclusion and answered accordingly.

The compiler requests all of the memory IT needs, but if the pool needs some additional memory for its purposes (pretty common), it will need to add that space itself. It's hard to imagine how it could be otherwise, I guess I would have thought that goes without saying. (And that rather proves that there is nothing that goes without saying.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:58:34 -0500

> But you cannot assume that the object is allocated as one big chunk. Bounds can be allocated at a different place. What would be X'Object_Address in that case?

The address of the real object, which is the bounds. (I'm using "object" in the Janus/Ada compiler sense and not in the Ada sense.) The only way I could make sense of the implementation requirements for Janus/Ada was to have every object be statically sized. If the Ada object is *not* statically sized, then the Janus/Ada object is a descriptor that provides access to that Ada object data.

Generally, one wants access to the statically sized object, as that provides access to everything else (there may be multiple levels if discriminant-dependent arrays are involved). That's not what 'Address is supposed to provide, so the address used internally to the compiler is the wrong answer in Ada terms, but it is the right answer for most uses (storage pools being an obvious example).

When one specifies 'Address in Janus/Ada, you are specifying the address of the statically allocated data. The rest of the object lives in some storage pool and it makes absolutely no sense to try to force that somewhere.

There's no sensible reason to expect 'Address to be implementation-independent; specifying the address of unconstrained arrays is nonsense unless you know that the same Ada compiler is creating the object you are accessing -- hardly a common case.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:19:38 -0500

> I don't care about techniques to implement meaningless stuff. It should be out, at least there must be a representation aspect for turning this mess off.

There is: Restriction No_Controlled_Types. - Randy

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:26:19 -0500

> Such implementations would automatically disqualify the compiler. Compiler-generated piecewise allocation is OK for the stack, not for user storage pools.

If someone wants to require contiguous allocation of objects, there should be a representation attribute to specify it. And there should not be nonsense restrictions on records with defaulted discriminants unless you specify that you require contiguous allocation. There is no good reason to need that for 99% of objects, why insist on a very expensive implementation of slices/unconstrained arrays unless it's required??

> No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

How else would you ensure that Finalize is always called on an allocated object? Unchecked_Deallocation need not be called on an allocated object. The Ada model is that Finalize will ALWAYS be called on every controlled object before the program ends; there are no "leaks" of finalization. Otherwise, one cannot depend on finalization for anything important; you would often leak resources (especially for simple kernels that don't try to free unreleased resources themselves).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:37:56 -0500

> 1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. I fail to see where that sort of thing might be even marginally useful.

The classic example of Finalization is file management on a simple kernel (I use CP/M as the example in my head). CP/M did not try to recover any resources on program exit, it was the program's responsibility to recover them all (or reboot after every run). If you had holes in finalization, you would easily leak files and since you could only open a limited number of them at a time, you could easily make a system non-responsive.

You get similar things on some embedded kernels.

If you only write programs that live in ROM and never, ever terminate, then you probably have different requirements. Most likely, you shouldn't be using Finalization at all (or at least not putting such objects in allocated things).

> 2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower

> scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

Pools are extremely low-level things that cannot be safe in any sense of the word. A badly designed pool will corrupt everything. Using a pool with the "wrong" access type generally has to be programmed for (as I answered earlier, if I assume anything about allocations, I check for violations and do something else.) And a pool can be used with many access types; many useful ones are.

Some of what you want is provided by the subpool mechanism, but it is even more complex at runtime, so it probably doesn't help you.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:45:28 -0500

> You can override the parent's Initialize and never call it. It is a plain primitive operation anybody can call any time any place. You can even call it before the object is fully initialized!

User calls on Initialize and Finalize have no special meaning; they're ignored for the purposes of language-defined finalization. The fact that they're normal routines and can be called by someone else means that some defensive programming is needed. That all happened because of the "scope reduction" of Ada 9x; the dedicated creation/finalization mechanism got dumped. Finalization was too important to lose completely, so Tucker cooked up the current much simpler mechanism in order to avoid the objections. It's not ideal for that reason -- but Finalize would still have been a normal subprogram that anyone could call (what else could it have been -- the mechanism of stream attributes could have been used instead). I don't think there is a way that one could have prevented user-defined calls to these routines (even if they had a special name, you still could have renamed an existing subprogram to the special name).

From: Simon Wright <simon@pushface.org>
Date: Tue, 28 Sep 2021 08:52:31 +0100

>> Deallocation is irrelevant. Finalization is called when objects are about to be destroyed, by any method.

> And no object may be destroyed unless deallocated.

Well, if it's important that an allocated object not be destroyed, don't allocate it from a storage pool that can go out of scope!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 28 Sep 2021 10:07:52 +0200

> Well, if it's important that an allocated object not be destroyed, don't allocate it from a storage pool that can go out of scope!

That was never the case.

The case is that an object allocated in a pool gets finalized because the access type (not the pool!) used to allocate the object goes out of the scope.

This makes no sense whatsoever.

Again, finalization must be tied with [logical] deallocation. Just like initialization is with allocation.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 28 Sep 2021 17:04:05 -0500

> Again, finalization must be tied with [logical] deallocation. Just like initialization is with allocation.

But it is. All of the objects allocated from an access type are logically deallocated when the access type goes out of scope (and the memory can be recovered). Remember that Ada was designed so that one never needs to use Unchecked_Deallocation.

I could see an unsafe language (like C) doing the sort of thing you suggest, but not Ada. Every object in Ada has a specific declaration point, initialization point, finalization point, and destruction point. There are no exceptions.

Code Flow Control

From: Kevin Chadwick <kevc3no4@gmail.com>
Subject: Code flow control
Date: Fri, 15 Oct 2021 08:08:42 -0700
Newsgroups: comp.lang.ada

Although surprised that pragma No_Exception_Propagation seems to prevent access to some exception information, I am happy with Ada's exception mechanism. I have read that exceptions should not be used for code flow.

For Ada after perusing various threads on this mailing list around best practice I am considering using exceptions locally but also have an in out variable for code flow control at the point of use. Is that the way with the caveat that it all depends on the task at hand?

In Go with vscode a static checker will warn if an error type variable is returned without a following if error utilisation (check usually of the form if err /= nil).

I have read that Spark has some kind of static analysis to achieve something similar as it forbids exceptions.

It is not the end of the world but is there any static analyser that could do similar for Ada. IOW save me some time or perhaps worse whenever I have simply omitted the check by accident, in haste or distraction.

I'm sure I could quickly write a shell script easily enough for a specific design as in if keyword appears once but not again within x lines then shout at me but I am wondering if I am missing a tool or better practice before I do so?

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 15 Oct 2021 19:48:06 +0200

> I have read that exceptions should not be used for code flow.

Some people reserve exceptions for signalling errors. I regard them as a way to handle "exceptional" situations, i.e. when the normal flow of control cannot continue. For example, in a deep recursive search, they are handy to stop the recursion and go back to top level when you have found what you were looking for. Some would disagree with that.

> I am considering using exceptions locally but also have an in out variable for code flow control at the point of use.

I definitely would prefer an exception, on the ground that you can omit the check, but you cannot ignore an exception.

> In Go with vscode a static checker will warn if an error type variable is returned without a following if error utilisation (check usually of the form if err /= nil).

An interesting idea for AdaControl, especially if you have some funding for it ;-)

From: Jeffrey R. Carter
Date: Fri, 15 Oct 2021 19:53:06 +0200

What you're talking about is result codes. Exceptions exist because of problems people encountered with result codes. Using result codes when you have exceptions is like using conditional go-tos when you have if statements.

So, yes, there is better practice. It's called exceptions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 15 Oct 2021 20:03:16 +0200

[...]

What I do wish is carefully designed exception contracts in Ada.

Named vs Anonymous Pointer Types

From: Simon Belmont <sbelmont700@gmail.com>
Subject: Is this legal?
Date: Sat, 16 Oct 2021 12:00:18 -0700
Newsgroups: comp.lang.ada

I'm trying to learn the 2012 changes to accessibility rules, e.g. aliased parameters, additional dynamics checks, and some

eliminated unnecessary typecasts. But I am also aware of the… fluid nature of GNATs correctness of implementing them, and the following situation seems dubious. In particular, when 'current' is an anonymous access type, it compiles without issue, but not when it's a named access type (or when explicitly converted to one). Does anyone know off hand which is the correct behavior?

Thanks

-sb

```
procedure Main is

   subtype str5 is string(1..5);
   type s5_ptr is access all str5;

   type T is
     record
       current : access str5;
       --current : s5_ptr; -- "aliased actual has
                          -- wrong accessibility"
       foo : aliased str5;
     end record;

   function F (y : aliased in out str5)
   return access str5 is
   begin
     return y'Access;
   end F;


   procedure P (x : in out T) is
   begin
     x.current := F(x.foo);
   end P;

  o : T := (current => null,  foo => "Hello");

begin
   P(o);
end Main;
```

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sun, 17 Oct 2021 01:35:22 -0700

For information, ObjectAda v.10.2 accepts both variants (in Ada 2012 mode).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 18 Oct 2021 22:50:52 -0500

>I'm trying to learn the 2012 changes to accessibility rules [...] Does anyone know off hand which is the correct behavior?

I can assure you that no one anywhere will *ever* know "off-hand" the correct behavior. :-) It takes quite a bit of looking to be sure.

...

...

> function F (y : aliased in out str5) return access str5 is

> begin

> return y'Access;

> end F;

This is always legal (we hope :-). There should be a static (or dynamic) check on Y when F is called that Y has an appropriate lifetime for the result. (I can't grok "accessibility", either. I always think in terms of lifetimes, and then try to translate to the wording.)

> procedure P (x : in out T) is

> begin

> x.current := F(x.foo);

> end P;

This should always be statically illegal. X here has the lifetime of P (as the actual lifetime is unknown). That's not long enough regardless of how you declare Current (since it's type is necessarily outside of P). There is no special accessibility rules for anonymous access components (unlike most other cases); they always have the accessibility (think lifetime) of the enclosing type.

[...]

Read a long UTF-8 File, Incrementally

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: How to read in a (long) UTF-8 file, incrementally?
Date: Tue, 2 Nov 2021 10:42:37 -0700
Newsgroups: comp.lang.ada

As I understand it, to work with Unicode text inside the program it is better to use the Wide_Wide (UTF-32) variants of everything.

Now, Unicode files usually are in UTF-8.

One solution is to read the entire file in one gulp to a String, then convert to Wide_Wide. This solution is not memory efficient, and it may not be possible in some tasks e.g. real time processing of lines of text.

If the files has lines, I guess we can also work line by line (Text_IO). But the text may not have lines. Can be a long XML object, for example.

So it should be possible to read a single UTF-8 character, right? Which might be 1, 2, 3, or 4 bytes long, so it must be read into a String, right? Or directly to Wide_Wide. Are there such functions?

Thanks a lot.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 2 Nov 2021 19:17:58 +0100

> So it should be possible to read a single UTF-8 character, right? Which might be 1, 2, 3, or 4 bytes long, so it must be read into a String, right? Or directly to Wide_Wide. Are there such functions?

You simply read a stream of Characters into a buffer. Never ever use Wide or Wide_Wide, they are useless. Inside the buffer you must have 4 Characters ahead unless the file end is reached. Usually you read until some separator like line end.

Then you call this:

http://www.dmitry-kazakov.de/ada/strings_edit.htm#Strings_Edit.UTF8.Get

That will give you a code point and advance the cursor to the next UTF-8 character.

However, normally, no text processing task needs that. Whatever you want to do, you can accomplish it using normal String operations and normal String-based data structures like maps and tables. You need not to care about any UTF-8 character boundaries ever.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Wed, 3 Nov 2021 00:43:02 -0700

> [...] Are there such functions?

There is a special library to process Unicode text, see https://github.com/AdaCore/VSS; 'contrib' directory contains VSS.Utils.File_IO package to load file into Virtual_String. However, attempting to load a whole file into the memory is usually a bad decision.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 3 Nov 2021 08:48:58 +0000

> As I understand it, to work with Unicode text inside the program it is better to use the Wide_Wide (UTF-32) variants of everything.

You can take a look at my simple lib: https://github.com/Lucretia/uca

It can read into a large string buffer. And can break it up into lines. There's no Unicode consistency checks.

The lib is a bit hacky, but seems to work for now. There's nothing more than what I've mentioned so far.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Thu, 4 Nov 2021 04:43:22 -0700

Great libraries, thanks.

It still seems to me that Wide_Wide_Character is useful. It allows to represent the character directly in the source code e.g.

```
if C = '±' then ...
```

And Wide_Wide_Character'Pos should give the codepoint.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Nov 2021 13:13:12 +0100

> if C = '±' then ...

If the source supports Unicode, it should do UTF-8 as well. So, you would write

if C = "±" then ...

where C is String.

> And Wide_Wide_Character'Pos should give the codepoint.

Yes, but you need no Wide_Wide to get an integer value and if your objective is Unicode categorization, that is too complicated for manual comparisons. Use a library function [generated from UnicodeData.txt] instead.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 4 Nov 2021 14:30:25 +0000

> And Wide_Wide_Character'Pos should give the codepoint.

Characters no longer exist as a thing as one can even be represented as multiple utf-32 code points.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Fri, 5 Nov 2021 03:56:42 -0700

You're alluding to combining characters?

From: Simon Wright <simon@pushface.org>
Date: Fri, 05 Nov 2021 19:55:33 +0000

> You're alluding to combining characters?

Fun & games on macOS[1]:

> $ GNAT_FILE_NAME_CASE_SENSITIVE=1 gnatmake -c p*.ads

> gcc -c páck3.ads

> páck3.ads:1:10: warning: file name does not match unit name, should be "páck3.ads"

>

> The reason for this apparently-bizarre message is that macOS takes the composed form (lowercase a acute) and converts it under the hood to what HFS+ insists on, the fully decomposed form (lowercase a, combining acute); thus the names are actually different even though they _look_ the same.

[1] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81114#c1

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Tue, 16 Nov 2021 03:55:05 -0800

I'm worried. I need the concept of character, for proper text processing. For example, I want to reference characters in a text file by their position. Any tips/references on how to deal with combining characters, or any other perturbing feature of Unicode, greatly appreciated.

(For me, a combining character is not a character, the combination is. Unicode agrees, right?)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 16 Nov 2021 13:36:00 +0100

> I'm worried. I need the concept of character, for proper text processing.

Simply ignore or reject decomposed characters.

> For example, I want to reference characters in a text file by their position.

That is no problem either. There are two alternatives:

1. Fixed font representation. Reduce everything to normal glyphs, use string position corresponding to the beginning of an UTF-8 sequence.
2. Proportional font. Use a graphical user interface like GTK. The GTK text buffer has a data type (iterator) to indicate a place in the buffer, e.g. when a selection happens. These iterators are consistent with the glyphs as rendered on the screen and you can convert between them and string position.

> (For me, a combining character is not a character, the combination is. Unicode agrees, right?)

No, Unicode disagrees, e.g. É can be composed from E and acute accent. But it is advised just to ignore all this nonsense and consider:

code point = character

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Tue, 16 Nov 2021 05:52:59 -0800

> Simply ignore or reject decomposed characters.

Brilliant!

> 1. Fixed font representation. Reduce everything to normal glyphs, use string position corresponding to the beginning of an UTF-8 sequence.

I am indeed resorting to byte position in UTF-8 files as the character position. Treating UTF-8 entities as the strings that they are:-)

(Not dealing with fonts nor graphics yet, just plain text.)

Thanks a lot.

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 16 Nov 2021 15:25:10 +0000

> [...] I want to reference characters in a text file by their position. [...]

You can't. The concept of character is dead, the new concept are grapheme clusters.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 16 Nov 2021 09:38:13 -0800

> (For me, a combining character is not a character, the combination is. Unicode agrees, right?)

You can use VSS and Grapheme_Cluster_Iterator to lookup for grapheme cluster at given position and to obtain position of the grapheme cluster in the string (as well as UTF-8/UTF-16 code units).

However, the concept of grapheme clusters doesn't handle special cases like tabulation stops; TAB is just a single grapheme cluster.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 16 Nov 2021 14:23:28 -0600

> Simply ignore or reject decomposed characters.

Unicode calls that "requiring Normalization Form C". ("Form D" is all decomposed characters.) You'll note that what Ada compilers do with text not in Normalization Form C is implementation-defined; in particular, a compiler could reject such text.

My understanding is that various Internet standards also require Normalization Form C. For instance, web pages are supposed to always be in that format. Whether browsers actually enforce that is unknown (they should enforce a lot of stuff about web pages, but generally just try to muddle through, which causes all kinds of security issues).

Happy Birthday, Ada!

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Subject: Happy Birthday, Ada!
Date: Fri, 10 Dec 2021 10:37:41 +0100
Newsgroups: comp.lang.ada

Born this day in 1815 and 1980. New version on this date in 2012.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 10 Dec 2021 11:30:11 +0000

Are new revisions always released on this date?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 10 Dec 2021 13:36:32 +0100

No. 83, 95, and ISO/IEC 8652:2007 were not published on Dec 10. Probably 2X won't be, either.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Fri, 10 Dec 2021 06:23:44 -0800

A list of dates Ada 93 .. Ada 2012 can be found here:
https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada%20Magica/Contents.html

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 10 Dec 2021 23:28:05 -0600

> Are new revisions always released on this date?

No. It was mostly a coincidence. We were told that the Ada 2012 Standard would be published in the December 15th batch of Standards. Joyce Tokar made a request to date it December 10th in honor of Ada's birthday; she got a response that ISO couldn't do that. When it actually was published (around December 15th), we found they had dated it December 10th as requested. Not sure when/where/how they changed their mind.

If it had been scheduled for, say, February 15th, such a request wouldn't make any sense. We wouldn't want to delay the Standard for months just so it can have a cool date.

Big_Integer Has a Limit of 300 Digits?

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Subject: Ada.Numerics.Big_Numbers.Big_Integer has a limit of 300 digits?
Date: Tue, 21 Dec 2021 21:57:08 -0800
Newsgroups: comp.lang.ada

I just started using the Big_Integer library that is a part of the 202X version of Ada.

It is repeatedly described as an "arbitrary precision library" that has user defined implementation.

I was under the impression that this library would be able to infinitely calculate numbers of any length, but there is clearly a default limit of 300 digits.

Is there any way to get rid of this problem?

[Example code omitted, as it is not referenced in later discussion. —arm]

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Wed, 22 Dec 2021 00:25:26 -0800

How did you determine the limit of 300 digits? I see nothing in your example, that would imply such a limit. Are you sure that you are not reaching a line length limit in Text_IO or maybe a limit in the Image attribute?

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 03:14:12 -0800

There is a limit

Bignum_Limit : constant := 200;

in System.Generic_Bignums body, function Normalize, lines 1100ff.

I do not see an implementation advice, implementation permission or implementation requirement about such limits in A.5.5, A.5.6.

But there is somewhere in the RM a paragraph stating that implementation may pose limits on certain features. I just cannot find the place in the RM.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 03:32:23 -0800

See RM 1.1.3(2).

I guess this limit is just a transient arbitrary limit until Ada 2022 is standardized.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 08:04:29 -0800

> Bignum_Limit : constant := 200;

RM 2.2(14) limits the line length and the length of lexical elements to 200.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 22 Dec 2021 17:01:56 +0000

What are you doing that requires that number of digits?

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Date: Wed, 22 Dec 2021 09:27:56 -0800

> What are you doing that requires that number of digits?

I am working on ProjectEuler.net problem number 48.

The questions asks you to sum the numbers n^n for (2 <= n <= 1000) and determine what the last ten digits of this number are.

Obviously, this is quite a trivial problem when using any arbitrary precision library.

I had incorrectly determined that 700^700 had 300 digits, in fact 700^700 = 3.7E1991.

However, my code strictly breaks when the loop range is set to 683, which 683^683 = 8.12E1935.

So, it is interesting that the Big_Integer type works with numbers of just under 2000 digit length despite Bignum_Limit : constant := 200.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 19:37:45 +0200

>> Bignum_Limit : constant := 200;

> RM 2.2(14) limits the line length and the length of lexical elements to 200.

To express it more clearly, RM 2.2(14) requires implementations to support lines and lexical elements of /at least/ 200 characters, but /allows/ implementations to support longer lines and lexical elements.

I'm not sure if GNAT supports more than 200 characters, though. And of course an Ada program that uses more than 200 characters may not be portable to compilers that support only 200.

But I don't see any direct logical connection to the number of digits that Big_Integers can support. While one cannot write a big-number literal longer than a line or longer than the maximum length of a lexical element, that should not directly limit the size of big-number values in computations.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 19:48:42 +0200

> I was under the impression that this library would be able to infinitely calculate numbers of any length,

I have the same impression (up to Storage_Error, of course).

How does it break? Some exception, or something else?

Mark Lorenzen suggested in an earlier post that the limit might be in the Big_Integer'Image function. The package Ada.Numerics.Big_Numbers.Big_Integers has some other output operations that you could try:

```
function To_String
(Arg : Valid_Big_Integer; ...) return String;
procedure Put_Image
(Buffer : ... ; Arg: in Valid_Big_Integer);
```

Of course those might be internally linked to the 'Image function and have the same possible limitation.

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Date: Wed, 22 Dec 2021 10:02:41 -0800

[...]

Niklas also gave me an epiphany as the exact error my program gives for the upper limit is

raised STORAGE_ERROR : Ada.Numerics.Big_Numbers.Big_Integers.Bignums.Normalize: big integer limit exceeded

I had thought that since the end of this error said big integer limit exceeded it was a problem with the library, but now I'm starting to think I need to get GNAT to allocated more memory for the program.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Wed, 22 Dec 2021 17:43:46 +0000

Does Ada's Big_Integer type work with modular ranged types?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 21:05:18 +0200

[...]

Modular types are not connected to Big_Integers, except that the particular problem you are trying to solve could be computed "mod 10**10" because it asks for only the last 10 digits. However, the Big_Integers package does not directly support computations "mod" something (perhaps this should be an extension in a later Ada standard, because such computations are quite common).

Using "mod 10**10" operations in solving the problem would limit the number of digits in all intermediate results drastically.

> Niklas also gave me an epiphany as the exact error my program gives for the upper limit is

>
Ada.Numerics.Big_Numbers.Big_Integers.Bignums.Normalize: big integer limit exceeded

[...] the very specific exception message ("big integer limit exceeded") suggests that this exception is not a typical Storage_Error (say, heap or stack exhaustion) but may indeed stem from exceeding some specific limit in the current Big_Integer implementation in GNAT.

The size of your problem, with only a few thousand digits, suggests that heap exhaustion is unlikely to happen. However, if the Big_Integer computations are internally recursive, and use stack-allocated local variables, stack overflow could happen, so the first thing to try would be to increase the stack size. Unfortunately, for the main subprogram that has to be done with some compiler or linker options which I don't recall now. (We should really extend pragma Storage_Size to apply also to the environment task, by specifying it for the main subprogram!)

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 22 Dec 2021 12:31:32 -0800

> I am working on ProjectEuler.net problem number 48. ...

The thing about Euler problems is they usually want you to figure out a clever math trick to get to the solution, rather than using brute calculation. In the case of this problem, you want to reduce all the intermediate results mod 1e10 (which fits in an int64 easily, though not quite in an int32). That gets rid of the need for bignums.

From: Simon Wright <simon@pushface.org>
Date: Wed, 22 Dec 2021 20:34:20 +0000

> There is a limit

> Bignum_Limit : constant := 200;

> in System.Generic_Bignums body, function Normalize, lines 1100ff.

This is the maximum length of a Digit_Vector, where

```
subtype SD is Interfaces.Unsigned_32;
-- Single length digit
type Digit_Vector is array
     (Length range <>) of SD;
-- Represent digits of a number
   (most significant digit first)
```

I think this should give a maximum value of ~10**2000.

I printed out sum'image'length; the last value before the exception was 1937.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 23 Dec 2021 08:31:11 +0000

Is mod overloadable?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 23 Dec 2021 09:54:34 +0100

It is as any operator.

P.S. For large numbers one needs rather full division than separate /, mod, rem.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Thu, 23 Dec 2021 03:41:13 -0800

> However, the Big_Integers package does not directly support computations "mod"

It does A.5.6(18/5).

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 23 Dec 2021 14:18:34 +0200

Yes, there is a "mod" operator for Big_Integer. My point was that there are no Big_Integer operations, such as multiplication, that are intrinsically modular in the same way as the operations for modular types are. So the only way to perform a modular multiplication of Big_Integers is to first multiply the numbers in the usual way, producing a possibly very large product, and then apply "mod" to reduce that product.

In my imperfect understanding, intrinsically modular big-number computations can be much more efficient than such post-computation applications of "mod", at least if the modulus is not itself a big number.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Thu, 23 Dec 2021 14:01:47 +0000

Yes, there are efficient algorithms for "x * y mod n" so almost all "big num" libraries provide a function to do it. Ada has the type system for the mod operation to be explicit in the type.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 23 Dec 2021 16:48:17 +0100

As it appears from the rest of the discussion that there is a limit in the implementation of the pkg, you could try using PragmARC. Unbounded_Numbers.Integers

https://github.com/jrcarter/PragmARC/blob/Ada-12/pragmarc-unbounded_numbers-integers.ad

where the implementation restricts a number to Integer'Last "digits" or the available heap memory, whichever is less.

Note that with recent versions of GNAT for 64-bit processors, the "digits" will be 64 bits.

Pure vs Preelaborate

From: Simon Belmont <sbelmont700@gmail.com>
Subject: Re: Ada Pure or Preelaborate or ? in Adare_net
Date: Mon, 3 Jan 2022 17:11:36 -0800
Newsgroups: comp.lang.ada

> I and a friend created an Ada network lib where, from the begining, we tried very hard to make It a Ada Pure.

> From the examples dir, the lib worked as expected (in gcc-10.2 gcc-11.2 and gcc-12). To our surprise, what most caught the attention of the group's friends was the fact that the lib was Ada Pure and if that was correct.

> For this reason, if really 'is' pure, not pure, preelaborate or what (?), pleeeeeeaaase, we ask the group's Ada Language Lawyers to help analyze and suggest modifications if necessary.

> link: https://gitlab.com/daresoft/network/adare_net/-/tree/202x

> for Ada version use 2012 and or 202x.

It seems to be mostly just a thin binding to a bunch of C functions, so the applicability of any Ada feature is mostly a moot point. The Ada compiler has no control or visibility into the C domain, so while on the one hand your packages are technically Pure, on the other hand the C functions can violate those "purity rules" all they want, which might be misleading to users expecting otherwise. You don't use 'Unchecked_Access either, but obviously that doesn't mean the C functions are somehow prevented from creating dangling pointers. Personally, I would have the interfaces reflect the reality of the actual behavior (which in the case of C code you don't control, is usually assume-the-worst).

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 4 Jan 2022 05:52:41 -0800

> For this reason, if really 'is' pure, not pure, preelaborate or what (?)

I recommend reading this: https://stackoverflow.com/questions/19353228/when-to-use-pragma-pure-preelaborate

If your units are declared as Pure, the compiler considers that they have no side effects and can decide to call the sub-programs only once and cache the result, or even not call the sub-program if the result is not used after.

From: Daniel Norte Moraes <danielcheagle@gmail.com>
Date: Wed, 5 Jan 2022 08:11:52 -0800

> It seems to be mostly just a thin binding to a bunch of C functions, so the applicability of any Ada feature is mostly a moot point. [...]

The C pointers are only created in c_initialize_socket.c (c_init_address()) and data pointed copied to an Ada array, and then immediately free by c part. This is the only time there is a dynamic allocation.

We managed to make libadare_net very close to 100% static allocation!

Because of this, there aren't dangling pointers.

There is still the problem of omitting the execution of subprograms by the compiler by pure packages.

Would 'preelaborate' solve this?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 5 Jan 2022 17:40:04 -0600

Yes. The permission to omit calls only applies to Pure (see 10.2.1(18/3)). http://www.ada-auth.org/standards/2xaarm/html/AA-10-2-1.html#p18. (I gave a reference to the Ada 2022 AARM, but this rule hasn't changed in spirit since it was introduced in Ada 95.)

From: Daniel Norte Moraes <danielcheagle@gmail.com>
Date: Thu, 6 Jan 2022 12:39:21 -0800

> Yes. The permission to omit calls only applies to Pure (see 10.2.1(18/3)).

Thanks!

We will change the packages in LibAdare_Net to 'preelaborate'. :-) and continue from here.