To receive the Ada User Journal, apply to Ada-Europe at:
http://www.ada-europe.org/**join**

ADA USER JOURNAL

Volume 42
Number 1
March 2021

Contents

	Page
Editorial Policy for *Ada User Journal*	2
Editorial	3
Quarterly News Digest	4
Conference Calendar	28
Forthcoming Events	36
Proceedings of the "HILT 2020 Workshop on Safe Languages and Technologies for Structured and Efficient Parallel and Distributed/Cloud Computing"	
M. Klemm, E. Quiñones, T. Taft, D. Ziegenbein, S. Royuela *"The OpenMP API for High Integrity Systems: Moving Responsibility from Users to Vendors"*	39
R. Wai *"XERIS/APEX: Hyperscaling with Ada"*	43
B. Kleinke *"Challenges and Lessons Learned Introducing an Evolving Open Source Technology into an Established Legacy Ada and C++ Program"*	48
K. Chard, Y. Babuji, A. Woodard, B. Clifford, Z. Li, M. Hategan, I. Foster, M. Wilde, D. S. Katz *"Extended Abstract: Productive Parallel Programming with Parsl"*	51
T. Taft, K. Chard, J. Munns, R. Wai *"Language Support for Parallel and Distributed Computing"*	55
Puzzle	
J. Barnes *"Cubes and Pyramids"*	59
Ada-Europe Associate Members (National Ada Organizations)	60
Ada-Europe Sponsors	Inside Back Cover

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

Preface by the News Editor	4
Ada-related Events	4
Ada-related Resources	7
Ada-related Tools	8
Ada Practice	13

[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,

The newsgroup has been very active in this period, so I must apologize for any threads with ellipsis in the part that you were finding most engaging, or if some of your answers are missing. On the bright side, c.l.a. is livelier than ever in recent memory, despite claims of NNTP being a thing of the past.

I begin my personal highlights with "Quick Inverse Square Root" [1] which, with the prompt of an Ada implementation, explores the fascinating origins of a numerical approximation algorithm found in an old C game engine and a key mysterious magic number. One contributor even reported a short thesis about it, which is also well worth the read if you find the topic interesting.

The newsgroup is not strange to strong opinions, and in this instance Randy Brukardt vehemently argued against raw arrays [2, 3] and interface usefulness [4], which led to involved debates on the appropriate levels of abstraction for certain data structures, orthogonality problems, and more. Coming from a compiler maker and ARG member, these opinions sure cannot leave one indifferent.

Finally, older (but, according to the thread, not simpler) times were revisited in a discussion about the possibility of adapting an Ada compiler for the 8051 chip [5]. Interesting points were made about its complexity and how useful can be a system with as little RAM as 64K.

Sincerely,
Alejandro R. Mosteo.

[1] "Quick Inverse Square Root", in Ada Practice.

[2] "Lower Bounds of Strings", in Ada Practice.

[3] "Array from Static Predicate on Enumerated Type", in Ada Practice

[4] "Simple Example on Interfaces", in Ada Practice.

[5] "Targeting the 8051 with Ada", in Ada Practice.

Ada-related Events

Ada at Online FOSDEM 2021 - 6-7 February 2021

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada at online FOSDEM 2021 - 6-7 February 2021
Date: Fri, 5 Feb 2021 06:58:50 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Hello everyone,

Some of you might be interested in the information below...

Dirk.Craeynest

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

Ada at online FOSDEM 2021 -
6-7 February 2021

#AdaFOSDEM #AdaProgramming #FOSDEM2020

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/21/210206-fosdem.html

"FOSDEM is a free event for software developers to meet, share ideas and collaborate. Every year, thousands of developers of free and open source software from all over the world gather at the event in Brussels. In 2021, they will gather online. No registration necessary." {quoted from https://fosdem.org/2021}

Although, as announced previously, there is no Ada Developer Room at FOSDEM 2021, we are pleased there will be some Ada-related content after all.

In short:

* AdaCore announced on Twitter: "Like previous years, we will participate in FOSDEM on Feb 6-7, 2021. AdaCore engineers will give two talks in the Safety and Open Source devroom! Check out the full blog post for more details.
* Egil Høvik pointed out on LinkedIn: "Someone did Advent of Code with a new language each day, one of which is Ada."
* There's a talk on Ada Lovelace and the first computer program.

The information in this message is also available at the URL above.

The dedicated FOSDEM pages mentioned there include links to the live stream and chat rooms for each presentation at the time of the event. Also useful is the link to the latest FOSDEM 2021 news, including info on attending a talk at FOSDEM 2021.

More about the presentations:

* "Adding contracts to the GCC GNAT Ada standard libraries" - to strengthen analysis provided by formal verification tools

  by Joffrey Huguet

  Saturday 6 February 2021 11:00-11:30

  Safety and Open Source devroom

  The guarantees provided by SPARK, an open-source formal proof tool for Ada, and its analysis are only as strong as the properties that were initially specified. In particular, use of third-party libraries or the Ada standard libraries may weaken the analysis, if the relevant properties of the library API are not specified. We progressively added contracts to some of the GCC GNAT Ada standard libraries to enable users to prove additional properties when using them, thus increasing the safety of their programs. In this talk, I will present the different levels of insurance those contracts can provide, from preventing some run-time errors to occur, to describing entirely their action.
* "Proving heap-manipulating programs with SPARK" - The SPARK open-source proof tool for Ada now supports verifying pointer-based algorithms

thanks to an ownership policy inspired by Rust

by Claire Dross

Saturday 6 February 2021 13:30-14:30

Safety and Open Source devroom

SPARK is an open-source tool for formal verification of the Ada language. Last year, support for pointers, aka access types, was added to SPARK. It works by enforcing an ownership policy somewhat similar to the one used in Rust. It ensures in particular that there is only one owner of a given data at all time, which can be used to modify it. One of the most complex parts for verification is the notion of borrowing. It allows to transfer the ownership of a part of a data-structure, but only for a limited time. Afterward ownership returns to the initial owner. In this talk, I will explain how this can be achieved and, in particular, how we can describe in the specification the relation between the borrower and the borrowed object at all times.

* "25 languages in 25 days"

by Peter Eisentraut

Sunday 7 February 2021 13:00-13:20

Lightning Talks

I did the Advent of Code 2020 with a different programming language every day, so instead of having to visit 25 developer rooms, you can just listen to me for my lightning summary.

* "Ada Lovelace and The Very First Computer Program"

by Steven Goodwin

Sunday 7 February 2021 17:00-17:40

Retrocomputing devroom

We all know that Ada Lovelace is credited as the first computer programmer. But what did she write? What did it do? And how does it work? We look at the program, its function, and break it down line-by-line so you can understand the origins of our entire industry. After all, it doesn't get any more retro than this! In this talk, developer, geek, and digital archaeologist, Steven Goodwin, breaks down the very first program ever written to explain what it does and how it works. He goes on to simulate it within a JavaScript version of Babbage's analytical engine, rewriting it piece-by-piece until it looks like modern code, and thereby demonstrate what features of current languages we now all take for granted. He finishes up with a discussion on the controversy surrounding her involvement in computing, aiming to answer the question once and for all - "Was she really the first programmer?"

(V20210204.1)

CfC Ada-Europe 2021 Virtual Conference - 31 Mar Deadline!

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC Ada-Europe 2021 Virtual Conference - 31 Mar deadline!
Date: Sun, 7 Feb 2021 18:04:14 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

*** UPDATED Call for Contributions - VIRTUAL EVENT ***

25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021)

7-11 June 2021, online

www.ada-europe.org/conference2021

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

*** Extended DEADLINE 31 MARCH 2021 AoE ***

#AEiC2021 ##AdaEurope AdaProgramming

News

- AEiC 2021 will be a virtual-only event.
- Deadline for Industrial Presentation outlines and Tutorial proposals is extended to 31 March 2021.

General Information

The 25 Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021 aka Ada-Europe 2021), initially scheduled to take place in Santander, Spain, will be held online from the 7th to the 11th of June, 2021. The conference schedule includes a technical program, vendor exhibition and parallel tutorials and workshops.

Despite the COVID-19 situation which led to the cancellation of the previous edition of the conference, there is a firm commitment to celebrate the 2021 edition in any case. The organizing committee estimates that the conditions for a safe in-person conference will not be met in June 2021. Consequently, the AEiC 2021 Conference will be a virtual-only event.

Schedule

14 January 2021: Submission of journal-track papers, and workshop proposals (CLOSED)

19 March 2021 Notification of acceptance for journal-track paper presentations and workshops 31 March 2021 Submission of Work-in-Progress (WiP) papers, industrial presentation outlines, and tutorial and invited presentation proposals

30 April 2021 Notification of acceptance for WiP papers, industrial presentation outlines, and tutorial and invited presentations

Topics

The conference is a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will have keynotes, Q&A sessions and discussions, and virtual 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:

- Design and Implementation of Real-Time and Embedded Systems: Real-Time Scheduling, Design Methods and Techniques, Architecture Modelling, HW/SW Co-Design, Reliability and Performance;
- Design and Implementation of Mixed-Criticality Systems: Scheduling Methods, Mixed-Criticality Architectures, Design Methods, Analysis Methods;
- Theory and Practice of High-Integrity Systems: Medium to Large-Scale Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities;
- Software Architectures for Reliable Systems: Design Patterns, Frameworks, Architecture-Centered Development, Component-based Design and Development;
- Methods and Techniques for Quality Software Development and Maintenance: Requirements Engineering, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues, Compilers, Libraries, Support Tools;
- Ada Language and Technologies: Compilation Issues, Runtimes, Ravenscar, Profiles, Distributed Systems, SPARK;
- Mainstream and Emerging Applications with Reliability Requirements: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Cloud Environments, Smart Energy Systems, Serious Games, etc;
- Achieving and Assuring Safety in Machine Learning Systems;

- Experience Reports in Reliable System Development: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics;
- Experiences with Ada: Reviews of the Ada 2012 language features, implementation and use issues, positioning in the market and in the software engineering curriculum, lessons learned on Ada Education and Training Activities with bearing on any of the conference topics.

Call for Journal-Track Papers

The journal-track papers submitted to the conference are full-length papers that must describe mature research work on the conference topics. They must be original and shall undergo anonymous peer review.

Accepted journal-track papers will get a presentation slot within a technical session of the conference and they will be published in an open-access special issue of the Journal of Systems Architecture (Q2 in the JCR and SJR ranks) with no additional costs to authors. The corresponding authors shall submit their work by 14 January 2021 via the Special Issue web page: https://www.journals.elsevier.com/journal-of-systems-architecture/call-for-papers/special-issue-on-reliable-software-technologies-aeic2021.

Submitted papers must follow the guidelines provided in the "Guide-for-Authors" of the JSA (https://www.elsevier.com/journals/journal-of-systems-architecture/1383-7621/guide-for-authors). In particular, JSA does not impose any restriction on the format or extension of the submissions.

Call for WiP-Track Papers

The Work-in-Progress papers (WiP-track) are short (4-page) papers describing evolving and early-stage ideas or new research directions. They must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 31 March 2021, via https://easychair.org/conferences/?conf=aeic2021, strictly in PDF and following the Ada User Journal style (http://www.ada-europe.org/auj/).

Authors of accepted WiP-track papers will get a presentation slot within a regular technical session of the conference and will also be requested to present a poster. The papers will be published in the Ada User Journal as part of the proceedings of the Conference. The conference is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar. The Ada User Journal is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Call for Industrial Presentations

The conference seeks industrial presentations that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, in the form of a short (one or two pages) abstract, by 31 March 2021, via https://easychair.org/conferences/?conf=aeic2021, strictly in PDF and following the Ada User Journal style (http://www.ada-europe.org/auj/).

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The abstract of the accepted contributions will be included in the conference booklet, and authors will get a presentation slot within a regular technical session of the conference.

These authors will also be invited to expand their contributions into articles for publication in the Ada User Journal, as part of the proceedings of the Industrial Program of the Conference.

Awards

Ada-Europe will offer an honorary award for the best presentation. All journal-track and industrial presentations are eligible.

Call for Invited Presentations

The invited presentations are intended to allow researchers to present paramount research results that are relevant to the conference attendees. There will be no publication associated to these presentations, which may include previously published works, relevant new tools, methods or techniques. The invited presentations will be allocated a presentation slot.

The Program Committee will select invited presentation proposals that may be submitted by e-mail to one of the Program Chairs as a one-page summary of the proposed presentation, along with the information and/or links required to show the relevance of the covered topic.

Call for Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference days. Workshop proposals should be submitted by e-mail to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the Ada User Journal.

Call for Exhibitors

The commercial exhibition will span the core days of the main conference. As an alternative to the traditional physical exhibition, a virtual room will be provided for exhibition activities. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

Organizing Committee

* Conference Chair

Michael González Harbour, Universidad de Cantabria, Spain
mgh at unican.es

* Program Chairs

Mario Aldea Rivas, Universidad de Cantabria, Spain
aldeam at unican.es

J. Javier Gutiérrez, Universidad de Cantabria, Spain
gutierjj at unican.es

* Work-in-Progress Chair

Kristoffer Nyborg Gregertsen, SINTEF Digital, Norway
kristoffer.gregertsen at sintef.no

* Tutorial & Workshop Chair

Jorge Garrido Balaguer, Universidad Politécnica de Madrid, Spain
jorge.garrido at upm.es

* Industrial Chair

Patricia Balbastre Betoret, Universitat Politècnica de València, Spain
patricia at ai2.upv.es

* 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

*** Program Committee

Mario Aldea Rivas, Univ. de Cantabria, ES

Johann Blieberger, Vienna Univ. of Technology, AT

Bernd Burgstaller, Yonsei Univ., KR

Daniela Cancila, CEA LIST, FR

António Casimiro, Univ. Lisboa, PT

Xiaotian Dai, University of York, UK

Juan A. de la Puente, Univ. Pol. de Madrid, ES

Barbara Gallina, Mälardalen Univ., SE

Marisol García Valls, Univ. Politècnica de València, ES

J. Javier Gutiérrez, Univ. de Cantabria, ES

Jérôme Hugues, CMU/SEI, USA

Patricia López Martínez, Univ. de Cantabria, ES

Lucía Lo Bello, DIEEI - Univ. degli Studi di Catania, ES

Kristina Lundqvist, Malardalen University, SE

Kristoffer Nyborg Gregertsen, SINTEF Digital, NO

Laurent Pautet, Telecom ParisTech, FR

Luís Miguel Pinho, CISTER/ISEP, PT

Jorge Real, Univ. Politècnica de València, ES

José Ruiz, AdaCore, FR

Sergio Sáez, Univ. Politècnica de València, ES

Frank Singhoff, Univ. de Bretagne Occidentale, FR

Tucker Taft, AdaCore, USA

Elena Troubitsyna, Åbo Akademi Uni., FI

Santiago Urueña, GMV, ES

Tullio Vardanega, Univ. of Padua, IT

*** Industrial Committee

Patricia Balbastre, Univ. Politècnica de València, ES

Dirk Craeynest, Ada-Belgium & KU Leuven, BE

Ahlan Marriott, White Elephant, CH

Maurizio Martignano, Spazio IT, IT

Silvia Mazzini, Intecs, IT

Laurent Rioux, Thales R&T, FR

Jean-Pierre Rosen, Adalog, FR

Previous Editions

Ada-Europe organizes annual international conferences since the early 80's. This is the 25th 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), and Warsaw, Poland ('19).

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 2021 Publicity Chair (aka Ada-Europe 2021)

Dirk.Craeynest@cs.kuleuven.be

* 25th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2021)

* June 7-11, 2021 * online event * www.ada-europe.org/conference2021 **

(V3.1)

Ada-related Resources

[Delta counts are from Feb 2nd to Apr 26th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Mon, 26 Apr 2021 22:51:21 +0100
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn:3_119 (+41) members	[1]
- Reddit: 6_426 (+1_931) members[1]	[2]
- Stack Overflow: 2_048 (+75) questions	[3]
- Freenode: 94 (+9) users	[4]
- Gitter: 75 (+9) people	[5]
- Telegram: 121 (+13) users	[6]
- Twitter: 43 (-17) tweeters	[7]
74 (-21) unique tweets	[7]

[1]Probably caused in part by confusion with the ADA cryptocurrency.

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

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

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

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

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

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

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

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Mon, 26 Apr 2021 22:51:21 +0100
To: Ada User Journal readership

Rosetta Code: 811 (+50) examples	[1]
38 (+1) developers	[2]
GitHub: 7631[1] (+8) developers	[3]
Sourceforge: 273 (-5) projects	[4]
Open Hub: 214 (+2) projects	[5]
Alire: 156 (+10) crates	[6]
Bitbucket: 89 (+1) repositories	[7]
Codelabs: 52 (=) repositories	[8]
AdaForge: 8 (=) repositories	[9]

[1]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, 26 Apr 2021 22:51:21 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. The IEEE ranking deltas are in regard to the 2019 edition, as it is updated annually. —arm]

- TIOBE Index: 30 (+2) 0.49% (+0.04%) [1]
- PYPL Index: 17 (+2) 0.8% (+0.15%) [2]
- IEEE Spectrum (general): 39 (+4) Score: 32.8 (+8.0) [3]
- IEEE Spectrum (embedded): 12 (+1) Score: 32.8 (+8.0) [3]

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

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

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

Ada-related Tools

HAC v.0.085

From: Gautier <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.085
Date: Fri, 1 Jan 2021 08:18:15 -0800
Newsgroups: comp.lang.ada

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

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

Source repository #1: https://sf.net/p/hacadacompiler/code/HEAD/tree/

Source repository #2: https://github.com/zertovitch/hac

* Improvements:

- HAC_Integer (internal name in HAC_Sys.Defs), i.e. HAC's Integer type, is now 64 bit.
- HAC_Float (i.e. `Real` in HAC programs) has now System.Max_Digits digits accuracy.
- Added range constraints, like: ` subtype Answer_Range is Character range 'a' .. 'z' `.
- Added membership test, like: ` x [not] in a .. b `.
- Several additions to HAC_Pack.
- Better I/O error handling.
- The whole system (Compiler and VM run-time) builds on both GNAT and ObjectAda64.

* Fixes ([hand_washing] all bugs stem from SmallAda [/hand_washing]):

- Recursive calls to main procedure were mistaken as calls to "standard" procedures in HAC_Pack.
- Block identification used main program's identifier instead of its nesting.
- EXIT statement on FOR loop implied stack corruption for several nested FOR loops.
- EXIT statements within IF statements didn't work properly.
- Priority levels in expressions were not conform to the Ada Reference Manual's. Most visible change: needless brackets can now be removed around logical expressions.

* Test suite: added new 19 programs to the 12 existing tests.

- The 19 source files are named exm/aoc/2020/aoc_2020_*.adb, solutions to the Advent of Code 2020 puzzles.

From: Gautier <gautier_niouzes@hotmail.com>
Date: Sun, 3 Jan 2021 02:53:40 -0800

> Maybe too early to ask, but is there an overview of what is implemented and not implemented?

Not too early at all! Here is an excerpt of doc/hac.txt which summarizes the current subset supported:

- You can define your own data types: enumerations, records, arrays (and every combination of records and arrays).

- Only constrained types are supported (unconstrained types are Ada-only types and not in the "Pascal subset" anyway).

- The "String" type (unconstrained) is implemented in a very limited way. So far you can only define fixed-sized constants, variables, types, record fields with it, like: Digitz: constant String (1..10) := "0123456789"; ... output them with Put, Put_Line, do comparisons and concatenations with expressions of the VString variable-length string type. For general string manipulation, the most practical way with the current versions of HAC is to use the VString's.

- There are no pointers (access types) and nor heap allocation, but you will be surprised how far you can go without pointers!

- Subprograms names cannot be overloaded, although some *predefined* subprograms, including operators, of the Standard or the HAC_Pack package, are overloaded many times, like "Put", "Get", "+", "&", ...

- Programmable modularity (packages or subprograms that you can "with") is not yet implemented.

- Generics are not yet implemented.

- Tasks are implemented, but not working yet.

- Small parts of the standard Ada library are available through the HAC_Pack package. You can see the currently available items in the specification, src/hac_pack.ads .

To get a "tangible" idea, you can look at the examples in the "exm" directory (run ../hac gallery.adb for a show), and the "exm/aoc/2020" directory. There is also stuff in "test", but programs there are not meaningful.

> Detail: all procedures need "with hac_pack; use hac_pack;"?

So far, yes. When modularity is implemented it will change...

From: Gautier <gautier_niouzes@hotmail.com>
Date: Thu, 7 Jan 2021 11:18:24 -0800

> Detail: all procedures need "with hac_pack; use hac_pack;"?

Actually not anymore, now (rev. #400+) you can write things like:

```
with HAC_Pack;

procedure Hello is
 procedure Prefixed is
 begin
  HAC_Pack.Put("Hello");
 end;
 procedure Using_Use is
  use HAC_Pack;
 begin
  Put(" World!");
 end;
begin
 Prefixed;
 Using_Use;
end;
```

:-)

LEA v.0.76

From: Gautier <gautier_niouzes@hotmail.com>
Subject: Ann: LEA v.0.76
Date: Fri, 1 Jan 2021 09:11:10 -0800
Newsgroups: comp.lang.ada

LEA is a Lightweight Editor for Ada

Web site: http://l-e-a.sf.net/

Source repository #1: https://sf.net/p/l-e-a/code/HEAD/tree/

Source repository #2: https://github.com/zertovitch/lea

Improvements:

- when no subwindow is open, Ctrl-W closes app
- Ctrl-H opens search & replace box
- new files have CR end-of-line's
- console I/O box scrolls to last line
- interaction with HAC: improved ergonomy of Text input boxes
- improved ergonomy of the "comment/uncomment selection" command

- embeds HAC (HAC Ada Compiler) v.0.085

Features:

- multi-document
- multiple undo's & redo's
- multi-line edit, rectangular selections
- color themes, easy to switch
- duplication of lines and selections
- syntax highlighting
- parenthesis matching
- bookmarks

Currently available on Windows.

Gtk or other implementations are possible: the LEA_Common[.*] packages are pure Ada, as well as HAC.

Enjoy!

GWindows Release, 01-Jan-2021

From: Gautier <gautier_niouzes@hotmail.com>
Subject: Ann: GWindows release, 01-Jan-2021
Date: Fri, 1 Jan 2021 12:24:57 -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 with the GNAT development system (could be made pure Ada with some effort).

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):

391: GWindows.Common_Controls. List_View: added Ensure_Visible.

387: (contrib) GWin_Util package: added Explorer_Context_Menu.

385: GWindows.Windows.MDI: added function Count_MDI_Children.

384: (contrib) Added GWin_Util package.

...and in gwindows\samples\drawing, a new demo: Game_of_Life_Interactive (you create life with mouse clicks :-)

GWindows Project site:

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

GWindows GitHub clone:

https://github.com/zertovitch/gwindows

Enjoy!

SweetAda 0.1h

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda 0.1h released
Date: Tue, 5 Jan 2021 09:37:13 -0800
Newsgroups: comp.lang.ada

I've just released SweetAda 0.1h.

SweetAda is lightweight development framework to create Ada systems on a wide range

of machines. Please refer to https://www.sweetada.org.

Release notes

- There is now a primitive SFP (Small-FootPrint) runtime, does nothing very interesting so far, only allows non-trivial exception declarations and floating-point validation; when I will implement the Secondary Stack, things should start to be far better
- RTS and PROFILE items are now lowercased, as well as RTS directory names
- RTS for MIPS* targets is tuned with -G0, you should use this in your target compiler setup
- RTS for SH* targets is tuned with -fno-leading-underscore, you should use this in your target compiler setup
- the Bits library unit now exposes BigEndian and LittleEndian static booleans
- new procedure Print (Interfaces.C.char) in Console library unit
- Tcl will be the default scripting language for complex tasks, it is strongly advised to install it in your machine (Windows users could download the tcltk.zip package) since script files will be gradually replaced, at least those too heavy for a shell
- as just said, the "createtxtrecord" tool in S/390 and the scripts for the creation of bootable PC floppy/hard disk images are now written quick-and-dirty in Tcl, but they should be widely usable and requires no external OS utilities support
- IDE driver sets LBA mode, and FAT (read-only) works with LBA logical sectors
- MBR library unit to recognize partitions (very minimal, only 1st partition detected)
- menu.bat now shows automatically a usage if an incorrect action was supplied
- libutils provides a createsymlink shell script to create symbolic (soft) links in an OS-transparent way, use it by referencing $(CREATESYMLINK) in the Makefiles; this substitutes a physical copy of files in non-Linux machines during subplatform-specific installation; however, in Windows machines it requires PowerShell elevation rights in order to avoid bloated warning messages, so adjust your OS settings; the good news are that is now possible to edit subplatform-specific files without lose your changes whenever you restart from scratch with a "createkernelcfg" build cycle
- Makefile cleanups, there are no scattered shell-dependent bloated constructs, except for the trivial ones, and they are now concentrated logically in few places; the build system should tolerate even spaces in pathnames (very bad practice, though)- delete unnecessary functions and variables in Makefiles
- reordering of gnat1 debug switches in Makefile.tc.in, corrected -gnatdt switch description
- reordering of configuration dump in Makefile
- reordering/deletion/tuning of compiler switches in various platforms
- new target MSP432P401R, very minimal, only blinks the on-board LED
- DE10-Lite NiosII target now performs stack setup and calls the low-level adainit function in startup.S, so that proper runtime elaboration happens
- AVR targets can now use aggregates (see explanation below)
- ArduinoUno does not specify the path to AVRDUDE executable, this is now delegated to the run script
- the S/390 target specifies a correct emulation mode in linking objects so that there are no more problems during processing
- typos, cosmetics and minor adjustments

Quick notes

As the release notes outlined, SweetAda should run on a bare 64-bit host system which supports, dependently on your target CPU setup, symbolic (soft) links and (optionally) Tcl/Tk. This is normal for Linux, Windows and OS X, so no concerns should arise. If you do not want to install the tcltk package I am providing from the SweetAda site, then download a package from your vendor, and specify the path to the tclsh executable in the top-level configuration.in.

The reason behind this is promptly understood: Tcl is a long-time HL language used in industrial automation and is currently used as a scripting tool in large applications like Xilinx Vivado, Altera Quartus and others. Also OpenOCD uses an embedded version that drives its user interface, so it is at least advisable to have a look, especially if you are working with SoC, embedded softcores or you are playing with JTAG programming on the bare metal.

To use SFP, please change settings in the top-level configuration.in:

RTS := sfp

PROFILE := sfp

USE_LIBADA := Y

Remember, you can change RTS at your will after a "make clean" or "menu.[sh|bat] clean".

Please do not rely on low-level layout of the filesystem hierarchies. When SFP runtime will be (hopefully) working, many files could be symlinks or separate units in order to switch between ZFP and SFPs. More precisely, low-level subprograms could start to declare private exceptions and interrupt-related RTS units, and this will prevent the use of a ZFP (which does NOT use anything from the compiler library, and this requires absolute care).

About aggregates in AVR targets. The problem is, aggregates could be Ada static RO objects, and so the back-end can legitimately allocate them in the .rodata section. Historically, .rodata section is quite often linked together with the .text, but unfortunately, AVR is an Harvard machine with separate address spaces, and the .rodata section should stay together with data sections in an executable image. Relocating Flash ROM .rodata in RAM during startup obviously is a no-op. Placing .rodata in RAM prevents the read-only behaviour, though. The ideal solutions could be to place .rodata in EEPROM, but this introduces a level of complexity that I see of little concernment so far. So the current decision is to place .rodata in RAM, and warn you about try to overwrite static data (it will require intimate knowledge of dereferencing machine-code objects, furthermore, objects are nevertheless hardly traceable, and this a very esotic, non-Ada, non-sense bad practice, so trying to do that implies hugely problems in other areas).

Last thing, as I've updated toolchains (without change timestamps), you are encouraged to re-download them, since exists the possibility that previous targets have problems in the GNAT/GCC wrappers, and do not emit compilation messages of dependent units during "brief", non-VERBOSE mode, as well as not generating Ada intermediate files nor assembler listing thereof. If you don't care about visual outputs or assembler analysis, simply ignore this.

As usual, download the three packages core, RTS and LibGGC (since many changes are system-wide), and please save your work before overwriting the filesystem.

Happy new year.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Thu, 7 Jan 2021 17:26:43 -0000

Good to see the MSP432!

I'm in the process (well, was ... must get back to it!) of updating the old MSP430 Ada system, now using the TI supplied GCC toolkit. This is a much easier build than the old one, and the official MSP430 backend has improved from the last time I looked at it a few years ago.

I must add taking a look at SweetAda to my task list...

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Date: Thu, 7 Jan 2021 10:33:12 -0800

I see you wrote about MSP430. Maybe you already know that MSP432 is a whole different thing, being an ARM-Cortex based chip. The MSP430 is instead a proprietary TI line of cores, which SweetAda does not support. Just to avoid misunderstandings -- apologize if I write something already clear to you.

That being said, I'll try to slowly work on MSP432. Next releases maybe will come with more peripherals I/O declarations to make the target barely usable. I use a MSP432P401R board, if you want to physically download code from the SweetAda environment via USB, you have to install OpenOCD, it's pretty simple by taking a look at the scripts.

Let me know and best regards,

SweetAda 0.2

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda 0.2 released
Date: Thu, 21 Jan 2021 09:24:20 -0800
Newsgroups: comp.lang.ada

I've just released SweetAda 0.2.

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

<u>Release notes</u>

- Makefile is now optimized and does not perform a bind phase every time; note, this requires an updated gnat-wrapper, please download a fresh copy of the toolchain
- Makefile "all-clean" target renamed as "distclean" (and so do all variables starting with "ALL_CLEAN...")
- Makefile: added GNATLS tool, deleted unnecessary variables, added .h dependencies in clibrary build, deleted C++ toolchain variables in Makefile.tc.in
- Makefile: double-quoted some file references which lead to errors if SweetAda lays in a path directory which contains spaces
- there is a new "share" directory, which contains various auxiliary files, in order to centralize sparse and/or duplicated files
- AVR ATmega328P targets specify now an emulation mode during linking objects so that the final ELF object has correct flags; this prevents, e.g., QEMU-AVR from exiting prematurely
- QEMU-AVR: startup.S #undef's __AVR_ENHANCED__ because QEMU isn't yet able to fully emulate ELPM instructions
- STM32F769I (disco) ARM-CortexM7, new target; only able to blink a LED (needs OpenOCD to communicate with the target from inside SweetAda)
- PC-x86-64 uses Tcl scripts for FD/HD booting in QEMU
- upgraded SPARCstation5 and DECstation5000, which missed the new $(SYMLINK) script
- Dreamcast target produces a CD-ROM image suitable to create a physical CDI
- S/390 can IPL SweetAda from DASD devices (thanks to Hercules' DASDLOAD -- you need it)
- S/390 createtxtrecord.tcl script now renamed as S360obj.tcl
- typos, cosmetics and minor adjustments

<u>Quick notes</u>

It is important to download also a fresh copy of the toolchain, because the changes will be triggered by an upgrade in the GNAT/GCC wrappers.

As usual, download the three packages core, RTS and LibGGC (since many changes are system-wide), and please save your work before overwriting the filesystem.

Ada Wav File Library v2.0.0

From: gustho...@gmail.com <gusthoff.ada@gmail.com>
Subject: Ann: Ada Wav File Library v2.0.0
Date: Thu, 7 Jan 2021 12:08:54 -0800
Newsgroups: comp.lang.ada

The Wav File Library v2.0.0, an open-source Ada library, has just been released:

https://github.com/Ada-Audio/audio_wavefiles/releases/tag/2.0.0

This library contains a Wav File Reader & Writer written in Ada 2012. It supports reading and writing of wav files, including the following features:

- Mono, stereo and multichannel audio.
- Audio samples with following bit depths: 16/24/32/64-bit PCM; 32/64-bit floating-point PCM

- Wave-Format-Extensible format (WAVE_FORMAT_EXTENSIBLE)

This library also includes support for PCM buffers in floating-point and fixed-point formats, as well as the automatic conversion between the data types used for the PCM buffer and the wavefile, which might have different formats (floating-point or fixed-point) or varying precision (e.g., 16 bits or 64 bits).

A detailed list of changes and new features can be found here:

https://github.com/Ada-Audio/audio_wavefiles/blob/2.0.0/CHANGELOG.md

A cookbook / tutorial can be found here:

https://github.com/Ada-Audio/audio_wavefiles/blob/2.0.0/cookbook/cookbook.md

Simple Components v4.55

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v
Date: Wed, 13 Jan 2021 13:01:54 +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 packages Universally_Unique_Identifiers and Universally_Unique_Identifiers.Edit were added to support UUID;
- Reboot procedure was added to the package GNAT.Sockets.Connection_State_Machine.ELV_MAX_Cube_Client.

Dotenv v1.0

From: Heziode <heziode@protonmail.com>
Subject: Dotenv - first release
Date: Fri, 22 Jan 2021 16:34:42 +0100
Newsgroups: comp.lang.ada

I have just released Dotenv: 1.0.0

Dotenv allows you to load environment variables from `.env` files.

For more information, please refer to: https://github.com/Heziode/ada-dotenv

UXStrings (UXS_20210207)

From: Blady <p.p11@orange.fr>
Subject: [ANN] UXStrings package available (UXS_20210207).
Date: Mon, 8 Feb 2021 12:22:12 +0100
Newsgroups: comp.lang.ada

UXStrings is now available on Github with the whole API implemented (version UXS_20210207 [1]).

The objectives are Unicode and dynamic length support for strings, those are closed to VSS [2] from AdaCore.

However, the UXStrings API is inspired from Ada.Strings.Unbounded in order to minimize adaptation work from existing Ada source codes. Gnoga and Zanyblue has been adapted to UXString with success, see Gnoga announcement [3].

This is a first implementation POC. UTF-8 encoding is chosen for internal representation. The Strings_Edit [4] library is used for UTF-8 encoding management. It has not been intensively tested but this implementation is to demonstrate the possible usages of UXString. A test program is also provided with some features demonstrated [5].

See readme [6] for full details.

Comments especially on specifications [7] are welcome and others too ;-)

Enjoy, Pascal.

[1] https://github.com/Blady-Com/UXStrings/releases/tag/UXS_20210207

[2] https://github.com/AdaCore/VSS

[3] https://sourceforge.net/p/gnoga/mailman/message/37199377/

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

[5] https://github.com/Blady-Com/UXStrings/blob/master/tests/test_uxstrings.adb

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

[7] https://github.com/Blady-Com/UXStrings/blob/master/src/uxstrings1.ads

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 11 Feb 2021 00:19:25 -0800

There is clearly a need here, given the number of implementations out there. I had also implemented GNATCOLL.Strings 4 years ago, with similar goals to yours:

- unicode support (via generic formal parameters and traits packages, so you can use UTF8, UTF16, ... internally)
- unbounded strings (with optional copy-on-write)
- task safety (using traits to choose what kind of counter to use)
- performance (small-string optimization: no memory alloc for strings of 18 characters or less)
- extended API (all missing subprograms from Ada.Strings.Unbounded)
- extensive testing

I must admit I am not sure why AdaCore chose to write VSS instead of improving one of their string implementations (ada.strings.unbounded, gnatcoll.strings,...) My initial idea had been that it would be possible to provide a nice generic package, highly configurable via traits, on top of which we could reimplement ada.strings.unbounded, ada.strings.bounded,...) but I left AdaCore before that could be accomplished.

I took a look at VSS and find the API confusing. Your API UXString is at least much clearer (if lacking doc at the moment :-)

I am hoping that the work on Alire (Ada package manager) will ultimately help us find one implementation that is good enough for everyone, and could ultimately become part of the language.

From: Blady <p.p11@orange.fr>
Date: Sat, 6 Mar 2021 19:13:24 +0100

UXStrings is now available with Alire (https://alire.ada.dev/crates/uxstrings), in your Alire project, just add UXStrings dependency:

```
% alr with uxstrings
```

Thus you can import the UXStrings package in your programs.

Pascal.

PS: for French readers, while referencing UXStrings on Alire, I make the opportunity to write a short howto with ALire:

https://blady.pagesperso-orange.fr/a_savoir.html#alire

AShell v1.0

From: Rod Kay <rodakay5@gmail.com>
Subject: Version 1.0 Release of aShell
Date: Tue, 16 Feb 2021 12:33:37 -0800
Newsgroups: comp.lang.ada

A component to aid in writing shell-like applications in Ada.

https://github.com/charlie5/aShell

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 17 Feb 2021 10:04:41 +0200

I suppose I could find out by looking more deeply into the component (which looks nice in the README), but I'm lazy today, so I ask: do you have a way of capturing the standard-error stream from a process, in addition to the standard-output stream?

From: Rod Kay <rodakay5@gmail.com>
Date: Thu, 18 Feb 2021 03:18:36 -0800

With the process 'Start' subprograms, you can provide your own input/output/error pipes. If not provided they default to the standard pipes.

```
function Start (Command : in String;
     Working_Directory : in String  := ".";
     Input : in Pipe    := Standard_Input;
     Output : in Pipe    := Standard_Output;
     Errors: in Pipe    := Standard_Error;
     Pipeline : in Boolean := False)
return Process;
```

The "Results_Of' function returns 'Command_Results' which provides access to data from both the Output_Pipe and the Error_Pipe.

In hindsight, this is not adequate. I will review over the weekend and attempt a better solution.

From: Jeffrey R. Carter
Date: Wed, 17 Feb 2021 12:05:17 +0100

Is this compiler and OS independent?

From: Rod Kay <rodakay5@gmail.com>
Date: Thu, 18 Feb 2021 03:29:35 -0800

Atm, the code uses Florist for 'POSIX' and one function from 'GNAT.OS_Lib'.

Florist appears to be gnat-specific ...

"FLORIST, an Ada application program interface for operating system services for use with the GNAT compiler and the Gnu Ada Runtime Library (GNARL)."

I have no means of testing on Windows. I hope that it may be possible to use with cygwin or a similar compatibility layer.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 18 Feb 2021 16:06:00 +0200

> Florist appears to be gnat-specific ...

Florist is an implementation of a standard for Ada-POSIX bindings, https://www.iso.org/standard/34354.html, so the Florist API should not be GNAT-specific.

However, the implementation of Florist may depend on the underlying system, including the Ada compiler and the OS.

Using the Florist API, rather than using GNAT libraries or OS functions directly, should increase the potential portability. Actual portability will depend on the existence of implementations, for the target system, of Florist or other realizations of the standard Ada-POSIX binding.

From: Mgr <mgrojo@gmail.com>
Date: Sat, 20 Feb 2021 23:58:37 +0100

> Florist is an implementation of a standard for Ada-POSIX bindings [...]

Some time ago, I gathered some information about compilers providing support of the Ada-POSIX standard for this Wikibooks article.

https://en.wikibooks.org/wiki/Ada_Programming/Platform/POSIX

From: Jeffrey R. Carter
Date: Thu, 18 Feb 2021 12:57:02 +0100

What is the advantage over using the compiler-supplied libraries to do these things?

From: Rod Kay <rodakay5@gmail.com>
Date: Fri, 19 Feb 2021 01:07:25 -0800

Ability to provide input data.

Ability to provide input/output/error pipes.

Ability to pipeline processes.

Several convenience functions to simplify the above.

Potential for increased portability.

AShell v1.1

From: Rod Kay <rodakay5@gmail.com>
Subject: Version 1.1 Release of aShell.
Date: Tue, 23 Feb 2021 15:39:42 -0800
Newsgroups: comp.lang.ada

- Factored out command code into a separate package.
- Simplified the specs.
- Added better error handling.
- Added several tests.
- Improvements for pipelines.

XNAdaLib 2021 Future Contents

From: Blady <p.p11@orange.fr>
Subject: XNAdaLib 2021 futur contents.
Date: Sun, 14 Mar 2021 10:39:31 +0100
Newsgroups: comp.lang.ada

I'm preparing XNAdaLib

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

2021 binaries for macOS Big Sur, the target content is:

- GTKAda 21.2
- GnatColl 21.2
- Florist latest
- AdaCurses 6.2
- Gate3 0.5c
- Components 4.55
- AICWL 3.24
- Zanyblue 1.4.0
- PragmARC latest
- GNOGA 1.6
- SparForte 2.4
- Alire 1.0.0
- Template Parser 21.2.

The GNAT compiler version should be Community 2021 when AdaCore will release it.

Is this packaging useful for you? Which packages are you using?

Feel free to send your wishes of missing Ada packages.

Thanks for your feedback, Pascal.

SparForte 2.4 Released

From: Ken Burtch <koburtch@gmail.com>
Subject: ANN: SparForte 2.4 released
Date: Sat, 20 Mar 2021 06:00:21 -0700
Newsgroups: comp.lang.ada

SparForte 2.4 Released.

SparForte is my Ada-based open source shell, programming language and web template engine. This release includes:

19 new features and examples

26 fixes (including the 1 from version 2.3.1)

5 changes

Version 2.4 has been tested on Linux, FreeBSD and Raspberry Pi.

The focus of this release was on command line and shell improvements.

The download links are available at the SparForte website. Please fill in the download poll so I know who is interested in the project.

https://www.sparforte.com/index.html

There is a blog article for the major features:

https://www.pegasoft.ca/coder/coder_january_2021.html

Not mentioned in the blog, --colour/--color will enable colour text and UTF-8 graphics in SparForte's messages. There is an equivalent pragma to enable it through a .sparforte_profile login file. It gives SparForte a more modern look.

I don't follow comp.lang.ada so follow up with any issues by email.

SparForte is a hobby and a volunteer project. I do not earn money from it.

Thanks and enjoy.

Status of AdaControl

From: J-P. Rosen <rosen@adalog.fr>
Subject: Status of AdaControl
Date: Fri, 26 Mar 2021 18:14:57 +0100
Newsgroups: comp.lang.ada

It's been a long time since the latest public release of AdaControl. But let me reassure my fellow users: AdaControl development and improvement never ceased, and Adalog is very active about it.

The latest wavefront versions are available on SourceForge (https://sourceforge.net/projects/adacontrol/) and GitHub (https://github.adalog.fr).

There is an issue with the community edition though: Last year, AdaCore separated the ASIS generator from the regular compiler - it is a new program called asis-gcc.

asis-gcc is part of a package called Asistools which is distributed only to Pro users. It is not part of the CE edition. This does not affect only AdaControl: gnatcheck has also been removed.

There is no problem for Pro users, and our own supported users receive updates regularly.

Debian and FSF-Gnat users, as well as users who stay with CE2019, will still be able to compile AdaControl, however it may crash sometimes due to not incorporating fixes for the latest issues that were discovered with the new features of AdaControl. These have been reported to AdaCore (and fixed).

However, we are not able to provide a compiled version for CE2020 users, which is what prevents us from making a complete release. We are investigating solutions for these CE users that we, at Adalog, want to continue to fully support without restrictions!

From: Simon Wright <simon@pushface.org>
Date: Fri, 26 Mar 2021 21:13:39 +0000

> There is an issue with the community edition though

FSF GCC 11 doesn't support ASIS either.

This will mean no gnatmetric, gnatpp, gnatstub, gnattest for macOS users, at least until I can escape the branch hell that's stopping me building libadalang!

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 26 Mar 2021 23:25:57 +0200

> There is no problem for Pro users, ...

Well, last time I asked, as a Pro user, AdaCore wanted extra lucre for the ASIS tools. So, a little problem...

Ada Practice

Re: Renames Usage

[Continues from “Renames Usage” in AUJ 41-4, December 2021, about the finer details of renamings. —arm]

From: Drpi <314@drpi.fr>
Subject: Re: renames usage
Date: Fri, 1 Jan 2021 13:39:39 +0100
Newsgroups: comp.lang.ada

Reading all the answers, I understand that:

```
X : Float renames Random (Seed);
```

is equivalent to :

```
X : constant Float := Random (Seed);
```

From: Jeffrey R. Carter
Date: Fri, 1 Jan 2021 15:46:39 +0100

Technically, the renames gives a name to the anonymous temporary object returned by the function. The constant declaration makes a constant copy of it. So they're equivalent, but not identical.

However, the compiler is free to optimize the copy away, and I'd be surprised if there are any compilers that don't (except GNAT with -O0).

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sat, 2 Jan 2021 17:00:13 +0100

Also remember that limited types do not permit copying, whether constant or not. Renaming avoids having to move an object at all:

[Example shortened by me. —arm]

```
task type Nail;  -- A limited type
type Nail_Reference is access Nail;

function Random_Pick return
Nail_Reference;

declare
   Choice : Nail renames Random_Pick.all;
```

From: Simon Wright <simon@pushface.org>
Date: Sat, 02 Jan 2021 17:22:27 +0000

Another reason for renaming [...] would be remembering a view conversion. [Example removed. —arm]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 2 Jan 2021 21:19:49 -0600

> [...] However, the compiler is free to optimize the copy away, and I'd be surprised if there are any compilers that don't (except GNAT with -O0).

In [the case of a scalar return], the "copy" is a register, and it would be hard (and pointless) to eliminate that. It's more interesting for a function that returns a composite object, and in that case your answer is correct. Note that you can tell if a copy is made if there is a controlled component in the object.

One thing we've learned in language design is that nothing is ever exactly equivalent to something else. There is always subtle differences. Typical programmers can ignore such stuff, but not language designers.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 1 Jan 2021 14:20:05 +0100

You must keep in mind that renaming ignores subtype constraints. So:

```
  X : Integer := -1;
  Y : Positive renames X;
  -- Let's fool ourselves
begin
  Put_Line ("A positive number " &
  Integer'Image (Y));
```

Will happily print "A positive number -1."

Quick Inverse Square Root

From: Matt Borchers <mattborchers@gmail.com>
Subject: Quick inverse square root
Date: Sat, 2 Jan 2021 14:26:30 -0800
Newsgroups: comp.lang.ada

I'm sure many of you have seen the Fast Inverse SquareRoot algorithm from the open source Quake III engine. I just encountered it a few days ago. Here it is, a bit reduced, from the original source:

```
//C code from Quake III engine
float Q_rsqrt( float number )
{
   long i;
   float x2, y;
   const float threehalfs = 1.5F;
      x2 = number * 0.5F;
   y = number;
   i = *(long *) &y;
   i = 0x5f3759df - ( i >> 1 );
   y = *(float *) &i;
   y = y * (threehalfs - (x2 * y * y));
   return y;
}
```

It is interesting how much clearer the Ada code version is:

```
with Interfaces; use Interfaces;
function QUICK_INVERSE_SQRT
   ( a : FLOAT ) return FLOAT is
    y : FLOAT := a;
    i : UNSIGNED_32;
    for i'Address use y'Address;
begin
    i := 16#5F3759DF# - shift_right( i, 1 );
    return y * (1.5 - (0.5 * a * y * y));
end QUICK_INVERSE_SQRT;
```

The magic hexadecimal number is calculated from the formula:

3/2 * 2**23 * (127 - mu) where mu is a constant close to 0.043.

My question is that I am trying to get this to work for Long_Float but I'm not having any luck. I would expect that everything should be the same in the algorithm except for the types (Float -> Long_Float and Unsigned_32 -> Unsigned_64) and the "magic" hexadecimal number that should be calculated from the same formula but adjusted for the Long_Float bit layout.

3/2 * 2**52 * (1023 - mu) where mu is the identical constant as used for Float case.

This doesn't seem to work and I haven't been able to find my error. I'm sure it is something silly. Does anybody have a suggestion?

A second question I have is how to make this a generic for any Floating point type. I can only think that I have to provide three things: not only the obvious Float type, but also the Unsigned type of the same size, as well as the hex constant.

```
generic
   type F is digits <>;
   type U is mod <>;
   magic : U;
function G_Q_INV_SQRT( a : F ) return
F;
```

I write the body like this:

```
function G_Q_INV_SQRT( a : F )
return F is
   y : F := a;
   i : U;
   for i'Address use y'Address;
begin
   i := magic - shift_right( i, 1 );
   return y * (1.5 - (0.5 * a * y * y));
end G_Q_INV_SQRT;
function QUICK_INVERSE_SQRT is
   new G_Q_INV_SQRT( FLOAT,
   UNSIGNED_32, 16#5F3759DF# );
```

This won't compile because the type U is not valid for the call to "shift_right". How do I overcome this obstacle?

Once that is overcome, is there a way I can eliminate having to pass in the unsigned type along with the floating point type? That seems like the programmer would require internal knowledge to make use of the generic. Any thoughts on how to get the compiler to compute the magic number in the generic at compile time?

From: Jeffrey R. Carter
Date: Sun, 3 Jan 2021 00:18:11 +0100

> This won't compile because the type U is not valid for the call to "shift_right". How do I overcome this obstacle?

Make it an explicit generic formal function parameter:

```
with function Shift_Right (...) return ...;
```

> Once that is overcome, is there a way I can eliminate having to pass in the unsigned type along with the floating point type?

You would want to make use of the attributes of floating point types in ARM A.5.3

http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-A-5-3.html

Whether these provide the information you need is another question. I don't see how you could declare the modular type in the generic.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sun, 3 Jan 2021 11:58:38 +0100

> [Original code]

This is not equivalent to C code, you have likely a typo error.

The formula you wrote above cannot be right. In effect, the factor y calculated from the exponent must be numerically the same for both float (IEEE 754 single-precision floating-point) and double (IEEE 754 single-precision floating-point). Which is apparently not. You should get the exponent multiplied by the same power of 2 as for float. For double, I make a wild guess, you should replace right shift by 1 with right shift by 30 = 32-2.

General notes.

1. C code relies on float being IEEE 754 single-precision floating-point number with endianness opposite to integer endianness numbers. The exponent must land in the integer's MSB. This is clearly non-portable.
2. The approximation is very crude. I am too lazy to estimate its precision within the intended range, which is what? [0, 1]?
3. Ergo, making it generic has no sense.
4. If you port it to Ada, add assertions validating endianness and floating-point format.

From: Matt Borchers
<mattborchers@gmail.com>
Date: Sun, 3 Jan 2021 14:31:15 -0800

Thank you Jeff and Dmitry. I have a generic functioning now.

Jeff,

Using attributes I was able to come up with a magic number using:

```
magic : constant U := U(3.0 / 2.0 *
2.0**(F'Machine_Mantissa - 1) *
(F(F'Machine_Emax - 1) - 0.043));
```

[...]

When people tell me that they use C for its low-level power and simplicity, like bit manipulations, and claim that other languages can't match C in that sense, I like to show them just how much better Ada can be -- aside from all the other benefits we all know. Eliminating the generic, I think the main algorithm is much clearer in the Ada version.

Here's my final code which seems to work well enough on my machine. The compiler required me to instantiate the generic with different names and then use renames for the function in the package specification.

```
with INTERFACES; use INTERFACES;
generic
   type F is digits <>;
   type U is mod <>;
   with function SHIFT_RIGHT( n : U;
            amount : NATURAL ) return U;
function G_QUICK_INVERSE_SQRT
            ( a : F ) return F;

function G_QUICK_INVERSE_SQRT
            ( a : F ) return F is
   magic : constant U := U(1.5 *
         2.0**(F'Machine_Mantissa - 1) *
         (F(F'Machine_Emax - 1) - 0.043));
   y : F := a;
   i : U;
   for i'Address use y'Address;
begin
   i := magic - shift_right( i, 1 );
   return y * (1.5 - (0.5 * a * y * y));
end G_QUICK_INVERSE_SQRT;

function QINVSQRT is
   new G_QUICK_INVERSE_SQRT(
         LONG_FLOAT,
         UNSIGNED_64, shift_right );
function QUICK_INVERSE_SQRT(
         a : LONG_FLOAT ) return
         LONG_FLOAT renames
         QINVSQRT;
function QINVSQRT is
   new G_QUICK_INVERSE_SQRT(
          FLOAT, UNSIGNED_32,
          shift_right );

function QUICK_INVERSE_SQRT(
          a : FLOAT ) return FLOAT
         renames QINVSQRT;
```

From: Jeffrey R. Carter
Date: Mon, 4 Jan 2021 00:47:13 +0100

Glad to have been of help.

Regarding the unsigned type, it seems this only works if F'Size = 32 or 64, so you could write versions that use Unsigned_32 and Unsigned_64, and then make your generic function do

```
if F'Size = 32 then
   return QISR32 (A);
elsif F'Size = 64 then
   return QISR64 (A);
else
   raise Program_Error with "F'Size must be
   32 or 64";
end if;
```

But I don't understand why this exists. In what way is it better than the (inverse) Sqrt operation of the FPU?

From: Matt Borchers
<mattborchers@gmail.com>
Date: Sun, 3 Jan 2021 19:50:03 -0800

[...]

> But I don't understand why this exists. In what way is it better than the (inverse) Sqrt operation of the FPU?

I mentioned first that this code comes from the Quake III engine. There must have been a purpose for it then or maybe it was never called but left in the source code. There are many videos about it on YouTube. I'm not really a low-level graphics guy, but I think it was intended to operate on the unit vector for intense graphics operations.

I think this algorithm would work on any floating point type with a bit layout similar to the IEEE-754 standard regardless of how many bits were allocated to the exponent and mantissa.

I don't have any personal use for it. It seemed like an easy example to show how Ada code can be simpler and just as powerful as C. I tried to turn it into a generic just as an exercise in trying to eliminate the modular type from the generic interface after I realized that two types were required that were related only in bit size. [...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 4 Jan 2021 13:13:00 +0100

> I haven't got the slightest idea for which range this function should be applied, but for sure not for the complete Float range.

It appears to be the Newton method with a heuristic used to choose the starting point. The description is here: https://en.wikipedia.org/wiki/Fast_inverse_square_root

It also mentions a hack for double precision IEEE 754 floats.

P.S. The method makes no sense to implement or use on modern hardware.

From: Egil H H <ehh.public@gmail.com>
Date: Mon, 4 Jan 2021 05:39:33 -0800

For anyone interested, there's a discussion on the algorithm in this paper:

https://cs.uwaterloo.ca/~m32rober/rsqrt.pdf

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Thu, 7 Jan 2021 17:49:32 -0000

> As computers get faster, storage gets larger, and code libraries get bigger, it is unfortunate that most programmers do not need to be as clever as they once were required to be.

> Thanks for finding and sharing the PDF paper! I'm amazed someone could write so many pages on this.

Having spent quite some time elsewhere getting sqrt down to a single clock cycle (throughput: 8 cycle latency) it doesn't surprise me at all. (The name Terje Mathisen comes to mind for assembly language implementations)

The odd coding (non use of union, strange use of intermediate variables) may well have been the result of compiler code generation limitations; the "better" form may have compiled to a few more instructions or run a little more slowly; not a good thing for a gamer on limited hardware!

Have you benchmarked the pretty Ada version against the original C ... or against a straightforward float operation on modern hardware?

Lower Bounds of Strings

From: Stephen Davies <joviangm@gmail.com>
Subject: Lower bounds of Strings
Date: Tue, 5 Jan 2021 03:04:31 -0800
Newsgroups: comp.lang.ada

I'm sure this must have been discussed before, but the issue doesn't seem to have been resolved and I think it makes Ada code look ugly and frankly reflects poorly on the language.

I'm referring to the fact that any subprogram with a String parameter, e.g. Expiration_Date, has to use something like Expiration_Date (Expiration_Date'First .. Expiration_Date'First + 1) to refer to the first two characters rather than simply saying Expiration_Date (1..2).

Not only is it ugly, but it's potentially dangerous if code uses the latter and works for ages until one day somebody passes a slice instead of a string starting at 1 (yes, compilers might generate warnings, but that doesn't negate the issue, imho).

There must be many possible solutions, without breaking compatibility for those very rare occasions where code actually makes use of the lower bound of a string.

e.g. Perhaps the following could be made legal and added to Standard:

```
subtype Mono_String is String (1 .. <>);
```

One question with this would be whether or not to allow procedure bodies to specify parameters as Mono_String when the corresponding procedure declaration uses String.

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 5 Jan 2021 12:24:44 +0000

> [...] it makes Ada code look ugly and frankly reflects poorly on the language.

Wrong. It highlights how poor programmers are, especially from other languages which love to hardcode numbers.

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 5 Jan 2021 21:08:55 -0600

IMHO, "String" shouldn't be an array at all. In a UTF-8 world, it makes little sense to index into a string - it would be expensive to do it based on characters (since they vary in size), and dangerous to do it based on octets (since you could get part of a character).

The only real solution is to never use String in the first place. A number of people are building UTF-8 abstractions to replace String, and I expect those to become common in the coming years.

Indeed, (as I've mentioned before) I would go further and abandon arrays altogether -- containers cover the same ground (or could easily) -- the vast complication of operators popping up much after type declarations, assignable slices, and supernull arrays all waste resources and cause oddities and dangers. It's a waste of time to fix arrays in Ada -- just don't use them.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 6 Jan 2021 10:13:06 +0100

> IMHO, "String" shouldn't be an array at all. [...]

It will not work. There are no useful integral operations defined on strings. It is like arguing that an image is not an array of pixels because you could distort objects in there when altering individual pixels.

> The only real solution is to never use String [...]

This will never happen. Ada standard library already has lots of integral operations defined on strings. They are practically never used. The UTF-8 (or whatever encoding) abstraction thing simply does not exist.

[...]

Array implementation is a fundamental building block of computing. That does not go either. Of course you could have two languages, one with arrays to implement containers and one without them for end users. But this is neither Ada philosophy nor a concept for any good universal-purpose language.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 Jan 2021 18:17:45 -0600

> [...] Array implementation is a fundamental building block of computing.

Surely. But one does not need the nonsense of requiring an underlying implementation (which traditional arrays do) in order to get that building block. You always talk about this in terms of an "interface", which is essentially the same idea. One cannot have any sort of non-contiguous or persistent arrays with the Ada interface, since operations like assigning into slices are impossible in such representations. One has to give those things up in order to have an "interface" rather than the concrete form for Ada arrays.

I prefer to not call the result an array, since an array implies a contiguous in-memory representation. Of course, some vectors will have such a representation, but that needs to be a requirement only for vectors used for interfacing. (And those should be used rarely.)

[...]

Sometimes, one has to step back and look at the bigger picture and not always at the way things have always been done. Arrays (at least as defined in Ada) have outlived their usefulness.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Thu, 14 Jan 2021 03:38:28 -0800

> I'm referring to the fact that any subprogram with a String parameter, e.g. Expiration_Date, has to use something like Expiration_Date (Expiration_Date'First .. Expiration_Date'First + 1) to refer to the first two characters rather than simply saying Expiration_Date (1..2).

I really do not see the problem here. If I want the first element, I write X(X'First). Where's the problem?

In his paper about model railroads, http://www.cs.uni.edu/~mccormic/RealTime/, John McCormick came to the conclusion that one of the reasons why Ada was so successful was the fact that indices had not to start with 0 resp. 1, i.e. they may bear meaning. In such cases, it is absolute nonsense to slide slices to the first index value.

Also for enumeration indices, sliding does not make sense.

So why is the bad habit dangerous to think that the first element must have index one (or zero)? For me, this is a non sequitur.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 14 Jan 2021 13:27:18 +0100

> Also for enumeration indices, sliding does not make sense.

Sliding does not make sense for any type of index.

Again, people are confusing indices (cardinal) with positions (ordinal). These are distinct concepts and different types. E.g. A'Length is an ordinal numeral and thus has the type Universal_Integer. A'First is a cardinal numeral and is of the index type.

> So why is the bad habit dangerous to think that the first element must have index one (or zero)? For me, this is a non sequitur.

The first element may have no index at all, e.g. the first element of a list, the first character read from the input stream etc.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Thu, 14 Jan 2021 05:31:57 -0800

> So why is the bad habit dangerous to think that the first element must have index one (or zero)? For me, this is a non sequitur.

Ah, what I really wanted to say: This is a bad and dangerous habit to think indices must start with 0 or 1.

From: Jeffrey R. Carter
Date: Thu, 14 Jan 2021 15:02:24 +0100

> Also for enumeration indices, sliding does not make sense.

The trouble is that this is not really discussing arrays. It's discussing sequences, implemented by arrays, such as String.

1-D arrays are often used to implement sequences. In arrays used as sequences, the indices are meaningless, and slicing, sliding, and sorting are often appropriate. As the indices are meaningless, it makes sense for them to be integers with a fixed lower bound of 1, since that is how we typically talk about positions in sequences. However, there are also many cases when it's useful to be able to have slices of sequences with a different lower bound, so remembering to use 'First is still important. Array types used as sequences are often unconstrained.

The other use of arrays (1- and multidimensional) is as maps. In arrays as maps, the indices are meaningful, and slicing, sliding, and sorting are usually inappropriate. Array types used as maps are usually constrained.

Ada's Vector containers are really variable-length sequences.

In designing a new language, it might be useful to keep these two concepts separate.

[...]

From: Stephen Davies <joviangm@gmail.com>
Date: Fri, 15 Jan 2021 02:24:40 -0800

> I really do not see the problem here. If I want the first element, I write X(X'First). Where's the problem?

```
Long_String_Name(1..2)
```

is much nicer than

```
Long_String_Name(
  Long_String_Name'First..
  Long_String_Name'First+1)
subtype Some_Range is Positive
range 4..5;
Some_String(Some_Range)
-- erroneous if Some_String'First/=1
```

I think the root of the problem is that Ada Strings almost always start at 1 (note that the functions in Ada.Strings.Fixed all return Strings that start at 1), so the cases when they don't are at best annoying, and potentially erroneous.

[...]

From: Jeffrey R. Carter
Date: Fri, 15 Jan 2021 12:48:25 +0100

> I think the root of the problem is that Ada Strings almost always start at 1

There are many cases where having String values with a lower bound other than 1 is more convenient, clearer, and less error prone than if all String values have a lower bound of 1. For example

```
loop
   exit when End_Of_File;
   declare
      Line : constant String := Get_Line;
   begin
      Idx := 0;
      loop
         Idx := Index (Line
              (Idx + 1 .. Line'Last), Pattern);
         exit when Idx = 0;
         Put_Line (Item => Idx'Image);
      end loop;
   end;
end loop;
```

where Index is Ada.Strings.Fixed.Index. Even without comments and descriptive

loop and block names, this is reasonably clear.

Compare that to a language where the slice slides to have a lower bound of 1 (because Index takes a String, which always has a lower bound of 1), and you'll see that it is more complex, less clear, and has more opportunities for error than current Ada.

A string, being a sequence, should usually have a lower bound of 1, but a decent language needs to also allow string values with other lower bounds. Maybe something like

```
type String_Base is array
      (Positive range <>) of Character;
subtype String is String_Base
      (Positive range 1 .. <>);
```

Slices would be String_Base, not String, and Index would take String_Base.

From: Stephen Davies <joviangm@gmail.com>
Date: Fri, 15 Jan 2021 06:00:43 -0800

> type String_Base is array (Positive range <>) of Character;

> subtype String is String_Base (Positive range 1 .. <>);

I wish it had been this way since the beginning. That way, in those rare instances where code is really using the variable lower-bound, the use of String_Base would make the intention clear. Alas, adopting this now would break that code.

From: Jeffrey R. Carter
Date: Fri, 15 Jan 2021 16:12:37 +0100

> I wish it had been this way since the beginning.

We have that now, with the substitutions

```
String_Base => String
String  => type S1 (Length : Natural) is
record
     Value : String (1 .. Length);
end record;
```

or

```
subtype S1 is String with
       Dynamic_Predicate => S1'First = 1;
```

From: Stephen Davies <joviangm@gmail.com>
Date: Fri, 15 Jan 2021 09:22:43 -0800

> subtype S1 is String with Dynamic_Predicate => S1'First = 1;

Like I said before, I want Sliding, not bounds checking. I guess most Usenet discussions eventually end up going around in circles.

From: Jeffrey R. Carter
Date: Fri, 15 Jan 2021 22:10:08 +0100

Then you would probably prefer the record version. Neither is perfect, but both, with appropriate conversion functions, give you the effect you want with current Ada.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sat, 16 Jan 2021 10:30:16 +0100

> Long_String_Name(1..2) is much nicer than

> Long_String_Name(Long_String_Name'First..Long_String_Name'First+1)

Avoid literals for indexing.

Of course, that makes them all the more popular. "On which side are you on 1 vs 0 for The First?" (Discussion starts...)

From: Stephen Davies <joviangm@gmail.com>
Date: Sat, 16 Jan 2021 05:13:49 -0800

> "On which side are you on 1 vs 0 for The First?"

I like that Ada gives the choice of "Positive range <>" or "Natural range <>".

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 18 Jan 2021 23:48:38 -0600

> Also, a Slide function [that returns the same string ensuring it is 1-based] does not work for "out" and "in out" parameters.

Thank god. Slices passed as in out parameters are the bane of the compiler-writers existence, and outside of types like String, have a very expensive implementation. On common machines like the x86, copying an arbitrary bit string from one location to another is not an easy operation to perform. (Remember, one can slice packed arrays, arrays of controlled objects, and other nasty cases. And with the sort of interface others here are proposing, you'd have to do it for various discontiguous representations, too.)

This way leads to madness -- at least of compiler implementers. ;-)

Record Initialisation Question

From: Drpi <314@drpi.fr>
Subject: Record initialisation question
Date: Sat, 9 Jan 2021 10:30:04 +0100
Newsgroups: comp.lang.ada

I'm working on a µP BSP [microprocessor board support package]. The boot sequence of this µP requires byte structures located in FLASH memory. For example:

```
type t_Dcd_Header is record
   Tag     : Unsigned_8  := 16#D2#;
   Length  : Unsigned_16 := 4; -- Length in
   -- byte of the DCD structure (this header
   -- included)
   Version : Unsigned_8  := 16#41#;
end record
  with Object_Size => 32,
       Bit_Order =>
System.Low_Order_First;
for t_Dcd_Header use record
   Tag     at 0 range  0 .. 7;
   Length  at 0 range  8 .. 23;
   Version at 0 range 24 .. 31;
end record;
```

The t_Dcd_Header is part of t_Dcd record.

The Length field of t_Dcd_Header must contain the length of t_Dcd.

```
Dcd : constant t_Dcd :=
 ( Dcd_Header => ( Length => ???,
   -- Length of Dcd
                 others => <>),
   ...
 );
```

Is there a way to automatically set Length ? Dcd goes in a dedicated .txt section.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sun, 10 Jan 2021 21:30:01 +0200

[Several possibilities are discussed involving static expressions, but the main obstacle turns out to be avoiding elaboration code. —arm]

>> Have you ensured that the construction of the Dcd object requires no elaboration code? Most Flash memories cannot be written in the same way as RAM, so even if that .txt section is not write-protected, normal RAM-oriented elaboration code would not be able to write into Flash.

> I'm aware of this (I'm an electronics guy). I'll add a "pragma No_Elaboration_Code_All;" when I'm ready.

Better add it now, because if you add it later, the compiler may then complain that it cannot implement the Dcd aggregate without elaboration code, and you will have to work around that somehow.

A good while ago, a colleague had a problem where a large constant array aggregate would require elaboration code if written in named form (Index => Value, Index => Value, ...), and it was necessary to write it in positional form (Value, Value, ...) to get rid of the elaboration code. It can be tricky, so it is better to be warned early of any problems.

From: Drpi <314@drpi.fr>
Date: Mon, 11 Jan 2021 18:46:34 +0100

I added "pragma No_Elaboration_Code_All;" to my code and... all records are rejected.

The boot data structure (in FLASH memory) is composed of several records. They are linked by their addresses. When a record contains an address, initializing it with a "non static number" value makes the compiler complain (with No_Elaboration_Code_All set).

You were right. I have to find a workaround.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 11 Jan 2021 22:58:57 +0200

> I added "pragma No_Elaboration_Code_All;" to my code and... all records are rejected.

Ah, too bad.

The problem is that "static" in Ada means "known at compile time", while addresses, although static in execution, are generally not known until link time. A case where assembly language is more powerful :-(

> I have to find a workaround.

If addresses are the only problem, and you are in control of the flash memory lay-out, you might be able to define static Ada constant expressions that compute ("predict") the addresses of every boot data structure record. But those expressions would need to use the sizes of the records, I think, and unfortunately the 'Size of a record type is not a static expression (IIRC), and that may hold also for the GNAT-specific 'Max_Size_In_Storage_Units.

From: Drpi <314@drpi.fr>
Date: Thu, 14 Jan 2021 14:07:29 +0100

> The problem is that "static" in Ada means "known at compile time", while addresses, although static in execution, are generally not known until link time. A case where assembly language is more powerful :-(

Or C :(

I use the manufacturer C code generated by their tool as reference. In C, initializing a structure element with an address is not a problem.

[...]

I can redefine the records with UInt32 instead of System.Address. The problem is: What is the expression to convert from Address to UInt32 without using a function?

From: Jeffrey R. Carter
Date: Thu, 14 Jan 2021 15:07:54 +0100

You can use an overlay (usually not recommended):

```
Addr : constant Address := ...;
U32  : constant Unsigned_32 with Import,
Convention => Ada, Address =>
Addr'Address;
```

You can also use an untagged union (also not usually recommended), which I would need to look up.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Thu, 14 Jan 2021 16:27:09 +0200

> [...] In C, initializing a structure element with an address is not a problem.

The C compiler emits a relocatable reference to the addressed object, and the linker replaces it with the absolute address. An Ada compiler should be able to do the same thing when the address of a statically allocated object is used to initialize another statically allocated object, assuming that the initialization expression is simple enough to require no run-time computation. Perhaps part of the reason why that does not happen is that System.Address is a private type, and might not be an integer type.

Do you (or someone) know if the C language standard guarantees that such initializations will be done by the linker, and not by the C start-up code that is analogous to Ada elaboration code?

[...]

But my suggestion did not involve such conversions: I assumed that you would be able to compute, using static universal-integer expressions, the addresses for all your flash objects, and use those directly in the record aggregates. This assumes that you are able to define the lay-out of all the stuff in the flash. You might then also specify the 'Address of each flash object, using those same universal-integer expressions.

Something like this (not tested with a compiler):

```
   Flash_Start : constant := 16#500#;
   Obj_A_Addr : constant := Flash_Start;
   Obj_B_Addr : constant := Obj_A_Addr +
16#234#;
   -- Here 16#234# is supposed to be the
size of Obj_A, so that
   -- Obj_B follows Obj_A in flash.

   Obj_A : constant Dcd_T := (
     Next => Obj_B_Addr,
     ...);

   for Obj_A'Address  use
   System.Storage_Elements.
   To_Address (Obj_A_Addr);
```

From: Paul Rubin
Date: Thu, 14 Jan 2021 08:59:04 -0800

> Do you (or someone) know if the C language standard guarantees that such initializations will be done by the linker, and not by the C start-up code that is analogous to Ada elaboration code?

I don't remember it being required by the standard, but I remember there was some pain in the standardization process trying to make those kinds of address initializations flexible while still being doable at link time. The original proposal had fancier capabilities than the final standard did, because during discussions it emerged that the fancy features couldn't straightforwardly be implemented with the linkers that people expected to use.

Specify Priority of Main Program

From: Simon Wright
<simon@pushface.org>
Subject: Specify priority of main program
Date: Sat, 23 Jan 2021 17:55:13 +0000
Newsgroups: comp.lang.ada

GNAT allows you to specify the main program's priority (actually, I suspect it'd allow it on any parameterless library-level procedure, but only the one actually used as main will count);

```
  procedure Main with Priority => 6 is
```

This is handy for embedded code where you don't want to waste the environment task's stack space but need to run that code at a non-default priority.

However, I can't see this use in the ARM; is it an extension?

If it's not a GNAT extension, what would the ARG view be likely to be for similar permission for Storage_Size (and Secondary_Stack_Size, but that is definitely a GNAT extension)?

From: Simon Wright
<simon@pushface.org>
Date: Sat, 23 Jan 2021 21:45:11 +0000

Found it now: ARM D.1(18).

This isn't mentioned in Annex J, Language Defined Aspects: (46),

"Priority of a task object or type, or priority of a protected object or type; the priority is not in the interrupt range. See D.1."

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 26 Jan 2021 20:52:50 -0600

[...]

>> If it's not a GNAT extension, what would the ARG view be likely to be for similar permission for Storage_Size (and Secondary_Stack_Size, but that is definitely a GNAT extension)?

I don't think the definition of Storage_Size would work out-of-the-box for a subprogram, since there wouldn't be an obvious place for it to get evaluated. So there's more work here than just slapping "for a subprogram" on the header. (Priority has to be static for a subprogram, and there is an additional rule explaining where it applies.)

But I don't see any other reason that Storage_Size shouldn't be allowed for a main subprogram. Probably it would take someone asking... :-)

Simple Example on Interfaces

From: Mario Blunk
<marioblunk.alere@gmail.com>
Subject: Simple example on interfaces
Date: Mon, 25 Jan 2021 08:08:05 -0800
Newsgroups: comp.lang.ada

I'm trying to solve a problem of multiple inheritance. It seems to me that an interface could be the solution although [interfaces are] still a mystery for me.

[A particular example omitted. The part of the conversation I have selected deals with general interface ideas, not depending on the particular example. —arm]

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 25 Jan 2021 23:06:09 +0100

[...]

Ada interface is a type that has interface and no implementation. (It is a silly idea inherited from Java.)

[...]

There exist various dirty tricks to emulate full multiple inheritance but no universal solution. If you really need full multiple inheritance, choose the most important path of implementations and make types along its proper types. Other paths if simple, could be tricked using

- Mix-in inheritance

- Generic packages to automate implementation of interfaces
- Memory pools to inject implementation

Nothing of these is good. They basically work only if the depths of the secondary inheritance paths are 1.

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 26 Jan 2021 10:37:12 +0100

> Ada interface is a type that has interface and no implementation. (It is a silly idea inherited from Java.)

To make it look a little less silly, think of it as a promise: a type that implements an interface promises to provide a certain number of operations.

Then you can define algorithms that work on any type that fulfills the promises.

To me, the big benefit of interfaces is that it is NOT inheritance; you say that your type provides some operations, without needing to classify it with an is-a relationship.

(I can hear screamings of pure-OO people who will not agree with me ;-)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 26 Jan 2021 11:25:37 +0100

> To make it look a little less silly, think of it as a promise

I agree. I meant that Ada 95 had that already:

```
type Interface is abstract tagged null
record;
```

There was no need to introduce it as a separate concept. I think the real reason was laziness. Vendors did not want to implement full multiple inheritance. Adding a simple constraint on the base types looked bad and also breached privacy:

```
type Is_It_Interface is abstract tagged
private;
private
   type Is_It_Interface is abstract tagged
   null record;
```

> To me, the big benefit of interfaces is that it is NOT inheritance; you say that your type provides some operations, without needing to classify it with an is-a relationship.

But you do. When you say that T provides F that in other words means T *is-a* member of a class that provides F. Interface is merely a formalization of that.

> (I can hear screamings of pure-OO people who will not agree with me ;-))

OO muddied a lot of water. To me things are quite pragmatic. How do I spell in the language the fact that Long_Integer is an integer? If Integer is an integer and Long_Integer is an integer can I write a program that works on integers? Can it be the *same* program for each instance of? Simple, natural questions.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Tue, 26 Jan 2021 03:15:01 -0800

> How do I spell in the language the fact that Long_Integer is an integer?

This is what generics are for (since Ada 83).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 26 Jan 2021 12:53:19 +0100

Right, generics is a form of polymorphism (static one). Generics have interfaces and these form classes.

[...]

P.S. Comparing generics to overloading, generics offer some re-use, and some degree of formalization at the cost of producing huge mess, while overloading does none.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Tue, 26 Jan 2021 08:46:05 -0800

> at the cost of producing huge mess

I know you don't like generics. I do not see a huge mess.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 26 Jan 2021 20:44:43 +0100

> I know you don't like generics. I do not see a huge mess.

When something goes wrong it is almost impossible to figure what. Contracts are mostly implicit. They are not enforced upon compilation. Instantiation errors nobody can really predict. On top of that is uncontrollable namespace pollution.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 26 Jan 2021 22:34:13 +0100

[...]

Generic packages and their formal parameters are organized in a directed acyclic graph like:

```
  A   D
/  \/ |
B  C  |
\  /  |
  E   |
  \  /
   F
```

rather than a tree. You want to instantiate the whole graph in a single shot. You do not want to manually specify constraints on generic formal parameters when some of them travel by several paths as D into F.

BTW, observe similarity with diamond/rhombus MI. That MI has some problems generics do not have is a big lie. But in my view generics are beyond salvation. The idea is inherently weakly typed. Ada's generic contracts are too loose to be safe and too rigid for usability of C++ templates.

From: Jeffrey R. Carter
Date: Mon, 25 Jan 2021 18:00:53 +0100

"IMHO, Interfaces are worthless."

Randy Brukardt

From: philip...@gmail.com <philip.munts@gmail.com>
Date: Tue, 26 Jan 2021 17:48:03 -0800

> "IMHO, Interfaces are worthless."

find interfaces to be extremely valuable for abstracting I/O devices. For example in my Linux Simple I/O Library, there is code equivalent to the following (the actual code is different, as I sucked a lot of common boilerplate for I/O device interfaces into a generic package that is instantiated for each data item type):

```
package GPIO is
  type Direction is (Input, Output);
  type PinInterface is interface;
  type Pin is access all PinInterface'Class;
  procedure Put(Self : PinInterface;
                       state : Boolean);
  function Get(Self : PinInterface)
  return Boolean;
end GPIO;
```

I've probably defined a dozen packages that implement GPIO pins using everything from Linux kernel services to web servers. Every one of them contains a function like this:

```
  function Create(...) return GPIO.Pin;
```

This allows code like the following:

```
  GPIO1 : GPIO.Pin :=
GPIO.libsimpleio.Create
(RaspberryPi.GPIO18, GPIO.Output);
  GPIO2 : GPIO.Pin := GPIO.HTTP.Create
("http://foo.munts.net", 5, GPIO.Output);
  GPIO3 : GPIO.Pin :=
GPIO.RemoteIO.Create
(server, 7, GPIO.Output);
```

This allows GPIO pins scattered far and near throughout the known universe to be treated exactly the same, even collected into an array or container.

I very seldom implement more than one interface in a type definition though, unless a single device has multiple sensors (temperature and humidity, for instance).

Microsoft's .Net uses this scheme pervasively, though I originally learned it in Ada and later applied the same thinking to .Net, Free Pascal, Java, Python, and C++ (and other languages).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 26 Jan 2021 21:36:53 -0600

> "IMHO, Interfaces are worthless."

> Randy Brukardt

To qualify that a bit, they're worthless to me (and I suspect, most people). For me, at least, OOP's benefits are mainly found in implementation inheritance, which is not available for Interfaces. You have to use abstract types to get those benefits.

For a single program, an interface doesn't buy anything, because it is very unlikely that you'll have more than one implementation of the interface in use. (Think the queue interface in Annex A.) So using dispatching just adds complication but no benefit; most likely you'll statically bind everything anyway.

Which pretty much leaves reusable code. Here, dispatching probably does have some benefit. But you can get similar benefits from generic units with formal derived type parameters. The problem is that interface dispatching is quite expensive (not just the indexing of single inheritance dispatching, but also some sort of lookup of the appropriate table). Whereas the generic solution does most of the binding at compile-time.

It may be my optimizer guru background, but indirect calls are pretty much unoptimizable. Ergo, the cost of dispatching is even worse than it appears on the surface, given that valuable optimizations like inlining, partial evaluation (currying), and all of the things that they enable aren't possible. So if the code performance matters, ultimately the interfaces will have to go. (Of course, if it *doesn't* matter, one shouldn't be warping a design for performance reasons. But it is *hard* to get rid of interfaces that are too expensive, so I think it makes most sense to be sparing with their use.)

Ultimately, I think one should only use interfaces IFF there is a clear reuse case where the substantial cost of dispatching is not a concern. For me, that is approximately never, but of course your mileage may vary.

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 27 Jan 2021 15:04:09 -0800

> Ultimately, I think one should only use interfaces IFF there is a clear reuse case where the substantial cost of dispatching is not a concern. For me, that is approximately never, but of course your mileage may vary.

It makes sense to use them in the internals of the compiler. Perhaps not a single-language compiler, but certainly a multilanguage one like GCC. An argument could be made for a single-language compiler in an environment like described in the DIANA reference-manual's rationale, where the DIANA-structure was meant to be passed around to things like pretty-printers and static-analyzers and code-generators.

You could make an argument that it would be useful for code-generators, too. I was contemplating using something like a hybrid of IEEE694 and 3AC last year... but that's a bit of a tangent.

https://standards.ieee.org/standard/694-1985.html

3AC = Three Address Code

GPS/GNAT Studio Code Completion Bug

From: John Perry <john.perry@usm.edu>
Subject: GPS/Gnat Studio: Code completion with other projects
Date: Sun, 31 Jan 2021 17:52:42 -0800
Newsgroups: comp.lang.ada

Suppose I've developed a package A, saved as a project. Now I'm working on package B. I make A available by specifying it in my gpr file, either as a with statement or by adding it to Source_Dirs. In package B I have the statement "with A;".

At this point, while I edit package B, GNAT Studio will code-complete any entity of package B, as well as any entity from the Ada standard library, but it won't code-complete entities from package A, such as A.Some_Feature.

How do I get GNAT Studio to do that?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 1 Feb 2021 08:51:58 +0100

It is a bug introduced in the latest version. Cross-referencing (it seems more than just auto-completion affected) across packages worked fine in earlier GPS versions.

From: Rod Kay <rodakay5@gmail.com>
Date: Tue, 2 Feb 2021 15:00:20 -0800

You might try this ...

To enable 'Find All References' => Append 'GPS.LSP.ADA_SUPPORT=no' to ~/.gnatstudio/traces.cfg

... it should help with finding references and refactoring.

From: Jérôme Haguet <j.haguet@cadwin.com>
Date: Fri, 12 Feb 2021 04:32:07 -0800

> Wow, that worked. Can you explain why? I don't see the connection at all. (I don't know what "GPS.LSP" means, either.)

You can find information in GNAT Studio Release Notes

https://docs.adacore.com/gps-docs/release_notes/build/singlehtml/index.html#document-relnotes_20

Specifying Only 'First of Array Index

From: Mehdi Saada <00120260a@gmail.com>
Subject: specifying only 'First of an index in an array
Date: Wed, 3 Feb 2021 09:47:14 -0800
Newsgroups: comp.lang.ada

Is there a way, on nominal or genetic array type definition (I mean in generic specifications), to ensure that Index_type'First is always the same, but that arrays can still grow?

Something like (certainly wrong): type my_type is array (Scalar_type range scalar_type'first .. <>) ?

That or I suppose I can wrap a function around that type and make it private to avoid range incompatibilities...

From: Jeffrey R. Carter
Date: Wed, 3 Feb 2021 22:45:17 +0100

This was discussed here recently referring specifically to strings.

Since these are sequences, the index should be numeric with a lower bound of 1.

Ada has had a way to do this since Ada 83:

```
type T_Base is array (Positive range <>) of
Element;
type T (Length : Natural) is record
   Value : T_Base (1 .. Length);
end record;
```

Ada 12 also adds the possibility of

```
subtype T is T_Base with
   Dynamic_Predicate => T'First = 1;
```

There is also the possibility of using a Vector for this.

The record has the advantage that sliding works, and the disadvantage that you have to put .Value in a lot of places.

The predicate has the advantage that it is an array type and objects can be indexed directly, and the disadvantage that sliding doesn't work.

Vectors have the advantage that the length can vary, and the disadvantages that slicing doesn't exist and conversions between Vector and T_Base are more complex than for the other forms.

Unreferenced Parameters

From: Simon Wright <simon@pushface.org>
Subject: Unreferenced parameters
Date: Wed, 03 Feb 2021 18:20:09 +0000
Newsgroups: comp.lang.ada

In gps-editors.ads:1492, in GNAT Studio, I have

```
overriding function Expand_Tabs
  (This   : Dummy_Editor_Buffer;
   Line   : Editable_Line_Type;
   Column : Character_Offset_Type)
 return Visible_Column_Type is (0);
```

and FSF GCC 10.1.0 says gps-editors.ads:1494:07: warning: formal parameter "Line" is not referenced gps-editors.ads:1495:07: warning: formal parameter "Column" is not referenced which is clearly the case (how does it know that it's OK not to reference This? it must check the context).

The compilation is set to treat warnings as errors (-gnatwe) so I need to suppress these warnings.

I could do so with pragma Warnings (Off, "formal*not referenced");

I have done so by renaming the parameters Dummy_Line, Dummy_Column.

But is there a way of using aspect or pragma Unreferenced? Putting pragma Unreferenced after the function definition doesn't work.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 3 Feb 2021 21:12:15 -0600

We (the ARG) recently added an allowance for aspect specifications on parameters and a few other constructs. The reason in part was because we didn't want to restrict where implementation-defined aspects can be placed, and the motivating case was aspect Unreferenced.

So I'd guess that you can put the aspect directly on the parameters in the usual way (but that may require a compiler not available yet; the change was approved in Sept [AI12-0395-1] and Oct [AI12-0398-1]). So, I'd expect the following to work (eventually):

```
overriding function Expand_Tabs
  (This   : Dummy_Editor_Buffer with
   Unreferenced;
   Line   : Editable_Line_Type with
   Unreferenced;
   Column : Character_Offset_Type with
   Unreferenced) return
   Visible_Column_Type is (0);
```

Array from Static Predicate on Enumerated Type

From: Matt Borchers <mattborchers@gmail.com>
Subject: array from static predicate on enumerated type
Date: Fri, 12 Mar 2021 12:49:27 -0800
Newsgroups: comp.lang.ada

Say, for example, I define a static predicate on a sub-type of an enumerated type, like:

```
type LETTERS is ( A, B, C, D, E, F, G, H, I ,
J, K );
subtype CURVED is LETTERS
  with Static_Predicate CURVED in
  B | C | D | G | J;
```

What I want is an array over CURVED (using CURVED as the index), but since attributes 'First and 'Last (and thus 'Range) is not allowed, this cannot be done.

Also, I am restricted in that the order of LETTERS cannot be rearranged.

Has anybody come up with a clever data structure to make sub-types with predicates easy and sensible for indexing (not iterating)?

I only need read access [...]

From: Jeffrey R. Carter
Date: Fri, 12 Mar 2021 23:16:29 +0100

It sounds as if you want a map, for which one of the map containers in the standard library would be appropriate.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 12 Mar 2021 23:41:53 +0100

> subtype CURVED is LETTERS

>   with Static_Predicate CURVED in B | C | D | G | J;

Do not use this thing, because its semantic is basically a lie as it violates contracts of other operations of the type, like 'Succ.

Using formal speak, CURVED is not substitutable for LETTERS in too many cases to be any useful.

This applies to any arbitrary constraints you could impose using a predicate. They break things. Do not ever consider them as an option.

From: Matt Borchers <mattborchers@gmail.com>
Date: Fri, 12 Mar 2021 18:06:22 -0800

I pretty much agree with Dmitry on this. The usefulness of this is very, very low without better support from the language itself. However, Dmitry, if programmers should not consider a feature of a language as an option for a solution, then it begs the question on the quality of the language, quality of the compiler, or questions the abilities of caretakers of Ada. Don't get me wrong, however, I think Ada is exceptional.

I thought I read that 'Pred and 'Succ do work as one would expect for the Predicated sub-type, but I did not try them as I did not need them.

I did read the entire rationale and 'First_Valid and 'Last_Valid would allow the programmer to create an array with a range that guarantees inclusion of all enumerated values of the statically predicated sub-type. But, this leaves holes in the array as wasted memory. My actual use case is hundreds of enumerated values and the sub-types have very few values each. Think of a case like a Unicode table where you might want to classify characters into small non-contiguous groups and these characters may be far apart from one another.

I do want a map or hash table, but in this case, I was hoping that Ada would handle the mapping for me such that I did not have to instantiate such a complexity for a simple example. I was a bit surprised after discovering Static_Predicate that the Ada language syntax was essentially useless in dealing with it in a consistent way.

I like the idea of creating non-contiguous enumerated sub-types. I've found that I often want to do it and must seriously consider design decisions like enumeration order that really should not be something that is that important to program design. I think that if the language lets you define them, then the rest of the supporting syntax of the language should also support them even if there is a small penalty of a double look-up through a mapping table.

I had a simple case many years ago with Ada 95, I think, when I was implementing a checkers game. I wanted an enumeration of 5 items for the piece that occupied a square.

```
type PIECE is ( EMPTY, RED, BLACK,
RED_KING, BLACK_KING );
p : PIECE;
```

This was a nice order because I could use the language syntax to determine if a piece was a King.

```
subtype KING is PIECE range
 RED_KING..BLACK_KING;
if p in KING then...
```

However, I had to write a function to determine if a piece was Red or Black and thus different calling syntax. The other order option was:

```
type PIECE is ( EMPTY, RED,
RED_KING, BLACK, BLACK_KING );
```

This order was nice because the language let me easily determine the Color of a piece.

```
subtype REDS is PIECE range
RED..RED_KING;
subtype BLACKS is PIECE range
BLACK..BLACK_KING;
if p in REDS then...
```

but I'd have to write a function to determine if a piece was a King and still different calling syntax.

Unfortunately, back then, the programmer couldn't have it both ways though it would've been very convenient. It appears that Static_Predicate solves this problem because "in" is updated to work with the Predicate. So if this works, why was it decided that the rest of the language syntax be inconsistent? Surely a map table would have sufficed with a slight performance penalty, but for the sake of language consistency you let the programmer decide. I can imagine an internally compiled map table would be much faster than the instantiation of the Map or Hash Container package.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 12 Mar 2021 22:55:51 -0600

>I do want a map or hash table, but in this case, I was hoping that Ada would handle the mapping for me

Ada is not some sort of magic wand. What you want requires a complex data structure, and using an array (as defined in Ada) for it is not practical (mainly because of the slice operation of which I've complained previously).

>...such that I did not have to instantiate such a complexity for a simple example.

Ada was designed to provide high-quality (that is, fast) code. If you want a language with a high degree of abstraction -- Ada isn't it. And in such a language, you wouldn't have arrays (in the Ada sense) at all - you would only have maps and sequences.

And if you think a single instance is "such complexity", I have no idea what you would want -- a map instance is simpler to write than an array type declaration (and *much* simpler under the covers). Do you also never use Unchecked_Deallocation?? It's harder to instantiate than an Ordered_Map.

>I was a bit surprised after discovering Static_Predicate that the Ada language syntax was essentially useless in dealing with it in a consistent way.

I was in favor of set constraints rather than Static_Predicates, mainly because of the value problems Dmitry commented on. But even those would have been illegal in arrays -- an array makes a lousy way to describe a map.

Anyway, subtypes with Static Predicates work for case statements, memberships, and for loops; they're only disallowed for arrays. I don't think anyone should be writing an array in a modern language (outside of interfacing to something outside of that language) - it's a mixed up data structure that only makes sense because of historical reasons.

> I like the idea of creating non-contiguous enumerated sub-types.

Static predicates do that fine. Just don't use them with obsolete data structures. :-)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 13 Mar 2021 09:04:51 +0100

[Bracketed comments in this post are from the author. —arm]

> I pretty much agree with Dmitry on this. [...]

Subtyping is a very difficult problem. When a new type is created by constraining [*] this necessarily breaks things.

Ada 83 was very careful to limit that to ranges and discriminant values. That breaks, sure, but the damage is minor and can be controlled [by the programmer]. In contrast, an arbitrary constraint [as well as arbitrary extension] is like a carpet bombing.

My view, as a programmer, is that features of type algebra [which subtyping by constraining is] must be carefully limited to enable massive language support in detection of substitutability issues at *compile* time. Features must be reasonably safe to use.

From: Matt Borchers <mattborchers@gmail.com>
Date: Mon, 15 Mar 2021 07:11:23 -0700

So, what I'm taking away from this discussion is that instantiating a Map is pretty much the best option when using a sub-type with a Static_Predicate to map a parent value to a sub-type.

[...]

It seems like the Ada community is always chasing higher adoption and better recognition of the Ada language. If the community truly wants this, then Ada needs to be accessible as a general purpose language with very few surprises. I evangelize for Ada when I can but I am of the opinion that language rules like these only frustrate people when they create seemingly inconsistent usability. There may be a good technical reason to break the behavior, but in this example and in my opinion, the technical excuse is not good enough when there is a very simple solution that the programmer should not have to implement. My 2 cents.

From: Matt Borchers <mattborchers@gmail.com>
Date: Mon, 15 Mar 2021 07:16:56 -0700

> Just don't use them with obsolete data structures. :-)

I can't tell if you are being facetious? If not, can you give me some reasons on why you think arrays are obsolete data structures? To me, they remain one of the basic building blocks of all programs.

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 15 Mar 2021 10:53:18 -0700

> I can't tell if you are being facetious? [...]

But they *AREN'T* maps, nor are they functions... despite the tendency to think of them as nails for your hammer (Array), this really isn't the case... and now that Ada has Ada.Containers.Indefinite_Ordered_Maps it really is an obsolete data-structure for mapping in most cases. (Exceptions exist for things like finite-state machines and virtual-machine instruction-sets where you're working with a uniform/near-uniform collection and/or things like embedded.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 16 Mar 2021 01:58:06 -0500

> can you give me some reasons on why you think arrays are obsolete data structures?

If you're talking *representation*, then surely arrays are the root of everything. But a general purpose programming language should hide representation issues as much as possible. For most uses, how a data structure is implemented is irrelevant; you want to ask for the fundamental data.structure that you need and let the implementation choose the best implementation to meet your needs.

And an array is not a fundamental data structure: those are bags and sequences and maps (and trees and graphs, but those aren't relevant here). Arrays have features of all of these, as well as some others -- they're not a fundamental data structure at all.

Moreover, Ada in particular merges in additional features that have little to do with data structures, and end up with a mixed up mess where one gets surprises from super-null arrays and arrays whose lower isn't 'First and holey arrays and other such nonsense.

For instance, the primary reason that Ada cannot have holey arrays is because of the slice (mis)feature, in particular because a slice can be assigned and (worse) passed as an in out parameter. If one has holey arrays, one also would expect to have holey slices (else the language would be quite inconsistent). But implementing a holey slice is problematic. For parameter passing, pretty much the only way to implement that would be to provide a call-back subprogram with every parameter that knows how to write each index of the slice. But that would be a classic distributed overhead -- it would be incurred for *every* array parameter since one can always create a holey slice of an array -- even of a type that is not itself holey. That would make passing strings and other arrays *much* more expensive.

[Example making the point omitted. —arm]

The point is that holey arrays are a massive can of worms, and it's impossible to have a consistent language if discontiguous subtypes exist. Tucker likes to say that sometimes language design is like a bump under a carpet -- you can move the bump around, but you can't get rid of it without ripping out the carpet and starting over (with a different language design). This is one of those cases.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 18 Mar 2021 12:15:30 +0200

[about "sparse" enumeration subtypes defined by static predicates, and arrays indexed by such subtypes]

>>> Nevertheless, it still feels like an unfinished feature as it is now.

>> It is not unfinished. It is irreparably broken.

> And this does not make for good advertising for Ada.

Matt, you should be aware that Dmitry has strong opinions about language and program design that are not shared by all Ada users and Ada proponents.

To be sure, Ada is showing some of its age. Updates of the Ada standards have made extensive additions to the language, while taking great pains to remain mostly upwards compatible, not only in syntax and semantics but also in wider usability goals such as remaining competitive for hard-real-time embedded systems and safety-critical systems where implementation overheads and implementation complexity must be held down. This inevitably means that new high-level features such as static predicates cannot always be fully orthogonal to other features of the language.

There have been suggestions and discussion here of an "Ada successor" language, and Dmitry in particular thinks that the type system should be completely overhauled for such a new language. Unfortunately there seems to be no demand from any large potential user group for such a language, or if there is demand, it is being satisfied mostly by new "grass-roots" languages such as Rust.

I have some hope that the swiftly growing scope and impact of malware and SW security breaches will prompt a major effort to develop computer systems, including programming languages, which are provably secure and incorruptible, and perhaps that will be an opportunity for an Ada successor language.

From: Jeffrey R. Carter
Date: Fri, 19 Mar 2021 01:49:39 +0100

> I wish I had the transcript from the Ada Group's discussions on this topic. It must have been a good one. Do they keep transcripts of their discussions? If so, does anybody know where to find them?

http://www.ada-auth.org/arg.html

You probably want ai05-0153-1 at

http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0153-1.txt?rev=1.15&raw=N

From: Matt Borchers <mattborchers@gmail.com>
Date: Mon, 22 Mar 2021 18:07:21 -0700

Thanks Jeff. This is going to take a while to get through and it is heavy reading. I had no idea this subject has been fermenting for 12+ years. However, in only the tiny portion I've read so far it seems a few commenters of high repute share some of my sentiments -- which only makes me 12 years late to the party of the losing side. :)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 22 Mar 2021 22:43:48 -0500

To get as complete as possible a picture of how some Ada feature came to be, you need to not only read the AI and especially its e-mail, but also the meeting minutes associated with that AI. We now have an index for that purpose on Ada-Auth.org, the Ada 2005 AI version is found at:

http://www.ada-auth.org/AI05-VOTING.HTML

Unfortunately, for Ada 2012, a lot of design occurred in unofficial phone meetings. No minutes were produced for those meetings, and so far as I know the only existing material is the notes I have kept on my hard disk. If I ever get some time, I want to get a version of those on-line so this sort of research can work usefully for Ada 2012. (Ideally in the format that the indexing tool can pick up and put into those indexes.)

Note that all three AI05-0153-x versions were involved, so it is useful to read all of them. (There also was some cross-AI discussions, which is probably beyond anyone's ability to find, at least for fun.)

Ada Style and "Early Return"

From: John Mccabe <john@mccabe.org.uk>
Subject: Ada and "early return" - opinion/practice question
Date: Mon, 15 Mar 2021 09:46:37 -0700
Newsgroups: comp.lang.ada

I hope this isn't a FAQ (it's hard to find relevant articles) but can someone guide me on the 'normal' treatment in Ada style of what appears to be referred to (by C/C++ programmers) as early-return.

For example, you have a C++ function (pseudo code sort of thing):

```
<sometype> fn(<some parameters>)
{
   if (<some undesirable condition 1>)
   {
      return <something bad happened 1>;
   }
   if (<some undesirable condition 2>)
   {
      return <something bad 2>;
   }
   if (<some undesirable condition 3>)
   {
      return <something bad 3>;
   }
   // Only get here if everything's good...
   <do some real stuff>
   return <something good>;
}
```

I've probably mentioned this before, but it's a long time since I used Ada in anger and I don't remember seeing stuff like that when I did use Ada a lot; does anyone write stuff like that in Ada?

When I first learnt to program properly it was using Pascal with, as I remember it, only one return from a function being allowed, so over the years I've mostly looked at positive conditions and indented stuff, pulling the stuff in the middle out into its own procedure or function where appropriate, but you see so many people defending this style in C/C++ that I wonder whether it really is defensible?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 15 Mar 2021 18:02:09 +0100

I see nothing wrong with it. [...]

P.S. The old mantra of structured programming was one entry, one exit. This is why some argued for single return while storing result code in a variable. Clearly adding a result variable would reduce readability rather than improve it.

P.P.S. One could debate exception vs. return code, but this is another story for another day.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 15 Mar 2021 10:31:27 -0700

Sometimes I write code that way, sometimes I have a Result variable that gets set along the way. The latter mostly when Result is a container of some sort, and parts of it get set at different points.

I would tend to use an exception for "something bad", but that depends on the overall design.

There are various maintenance issues on both sides; the summary is "editing existing code is a pain" :(.

From: Jeffrey R. Carter
Date: Mon, 15 Mar 2021 19:37:02 +0100

[In reply to the original post. —arm]

Other than the use of exceptions rather than a return code, this is a standard idiom in Ada. It's much easier to read and understand than the Pascal approach, just as a "loop and a half" is much clearer with an exit than the Pascal approach.

I seem to recall Robert Dewar arguing for this style on here many years ago.

From: John Mccabe <john@mccabe.org.uk>
Date: Mon, 15 Mar 2021 11:54:01 -0700

> I seem to recall Robert Dewar arguing for this style on here many years ago.

From what I remember of Robert (RIP), I suspect he probably argued against it at some point as well, depending on who he was arguing with :-)

Elaboration Code, Aggregates

From: Simon Wright <simon@pushface.org>
Subject: Elaboration code, aggregates
Date: Sun, 28 Mar 2021 20:41:25 +0100
Newsgroups: comp.lang.ada

In June 2020, Luke A. Guest was having trouble with getting the compiler to place constant data into the data section without elaboration code.

https://groups.google.com/g/comp.lang.ada/c/B2NA-qjCJuM/m/4ykywZWZAgAJ

Can be found as "Putting Data in the .data Section", in AUJ 41-2, June 2020. —arm]

During preliminary work for FSF GCC 11, I found that this ARM interrupt vector (which used to compile happily without needing elaboration code) no longer would:

https://github.com/simonjwright/cortex-gnat-rts/blob/master/stm32f4/adainclude/startup.adb#L231

[Example removed as it is equivalent to the one following. —arm]

and Arduino Due clock startup didn't:

https://github.com/simonjwright/cortex-gnat-rts/blob/master/arduino-due/adainclude/startup-set_up_clock.adb#L48

```
PMC_Periph.CKGR_MOR :=
(KEY => 16#37#,
 MOSCXTEN => 1, -- main crystal
                       -- oscillator enable
 MOSCRCEN => 1, -- main on-chip rc osc.
                       -- enable
 MOSCXTST => 8, -- startup time
 others   => <>);
```

On investigating, it turns out that FSF GCC 11 **AND** GNAT CE 2020 have lost the ability to assign aggregates as a whole; instead, they assign the record components one-by-one.

The reason for the Arduino Due failure is that the PMC hardware requires that each write to the CKGR_MOR register contain that value of KEY! so the sequence is

read the register (KEY is always returned as 0)
overwrite the KEY field
write the register back
read the register, KEY is 0
overwrite the MOSCXTEN field
write the register back, KEY is 0 so inoperative
etc (including the 'others => <>' components).

Bug report raised:

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=99802

From: Andreas Zeurcher <zuercher_andreas@outlook.com>
Date: Mon, 29 Mar 2021 11:49:06 -0700

Turn-around time from submission to general-availability of a released fix can be quite long in FSF GNAT or Community Edition. (Paid-support for GNAT Pro at AdaCore can be more prompt, I hear.)

From: Simon Wright <simon@pushface.org>
Date: Mon, 29 Mar 2021 20:03:42 +0100

Maybe, but this is accepted as a regression and Eric is on it! :impressed:

Paid support can be very prompt. We were at the stage where our Systems Engineer couldn't accept a compiler change, so wavefronts wouldn't have helped, but workrounds were indeed prompt.

From: Simon Wright <simon@pushface.org>
Date: Tue, 30 Mar 2021 08:08:34 +0100

Now fixed on GCC mainline.

Cross-compiler for Embedded Linux on ARMv7?

From: John Mccabe <john@nospam.mccabe.org.uk>
Subject: Are there any cross-compiler for Embedded Linux on ARMv7?
Date: Mon, 29 Mar 2021 17:16:42 -0000
Newsgroups: comp.lang.ada

Kind of as it says in the subject; I'm aware there's a GNAT Pro release that seems to target Embedded Linux on ARM, but are there any others?

I'm assuming the GNAT offering covers ARMv7 on the basis their bare-metal one packaged in the Community Edition does, but maybe it doesn't!

I saw some information on a PTC ApexAda one but what I read gives the impression it may be ARMv8 only, maybe not though!

If anyone knows more about this, any info they can give me would be very much appreciated; at this point I'm particularly interested in ARM A9 support, and at least Ada 2005, preferably 2012.

Also, does anyone know what AdaCore's like (or any other vendors, for that matter) if you ask for pricing/evaluation? We've been using C++ at work for ages, but I'm quite interested in seeing whether it would be at all feasible to move, at least partly, to Ada because C++ gets on my nerves :-) Sadly though, as we're busy and it would be an "on the side" evaluation, I've not got much time to 'play' with it, so the duration would be pretty much be open-ended, and I could do without people hassling me every few weeks to buy their products when the chances are I've managed about 10 minutes with it between calls...

Hope you don't mind me asking here; I know there are some great guys from various vendors here, so...

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 29 Mar 2021 20:26:47 +0200

We are using GNAT Pro cross compiler with Yokto and Debian, though I presume it will work with any distribution.

You need no evaluation. Simply install Debian, Ubuntu or Fedora on a reasonable ARM board 2GB or more. Use the native GNAT FSF compiler there to build your executable. Transfer it to the target board. Enjoy.

Once you are ready, go and buy GNAT Pro.

From: John Mccabe <john@nospam.mccabe.org.uk>
Date: Mon, 29 Mar 2021 21:06:32 -0000

Thanks for that info Dmitry. We're using Petalinux on custom hardware with a Xilinx Zynq-7000 (dual-core ARM A9), so it would be nice to run it on the real thing to work out how we'd deal with some of the FPGA interfaces and so on, if we were to purchase.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 29 Mar 2021 23:40:46 +0200

If you plan to run Linux there I see no reason why you could not use the native ARM compiler for evaluation. A cross compiler would change little or nothing in that case.

We are using a cross compiler for our custom target boards because it can be hosted on a powerful x86 machine instead of a sluggish ARM which also tends to crash under load or freeze when it goes into the swap.

Otherwise, nothing changes. We can perfectly well compile everything using GNAT FSF on an ODROID-XU4. It would only take a week instead of a day to build…

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 30 Mar 2021 20:12:36 +0200

> The Zynq-7000 we're using is a dual-core ARM A9 (as I mentioned) running at between 866MHz. As far as I can see the ODROID XU4 has quad-A15s at 2GHz + quad-A7s at 1.4GHz, with 2GB RAM. So, if you imagine the "week instead of a day" thing, then take into account the dual-core vs 8-core, 866MHz vs 2.0GHz/1.4GHz, 1.0GB vs 2.0GB, and RAM filesystem (ok, admittedly we have got 4GB flash on there, but...), perhaps a native ARM compiler isn't going to be a very effective evaluation tool :-)

One of our target boards is only 512M RAM single core.

The trick is to build on ODROID, but to run on the target.

Our code basis is huge, which is why it takes so long to build. For a sizable project ODROID is OK. When I compile my private stuff it takes 12+ hours to recompile everything on a Raspberry Pi 3, and only 3-4 on an ODROID.

The main problem is to figure out the gprbuild -j<n> switch. -j0 will likely run you into the swap with 8 kernels and many generics. ARM Linux becomes unstable when swapping.

If you invest in writing a good mock for your hardware, you could develop and test mostly on an x86. Only the integration tests would require building on the ODROID and running on the target.

From: Andreas Zeurcher <zuercher_andreas@outlook.com>
Date: Mon, 29 Mar 2021 11:46:20 -0700

> Also, does anyone know what AdaCore's like (or any other vendors, for that matter) if you ask for pricing/evaluation?

The sales staff is pleasant to deal with, but you might get sticker shock at the prices that they charge for non-GPLed supported products. As far as evaluation, I think that you are looking at it with the GPLed Community Edition, that is something that you should ask the salesman to see whether there is in fact any evaluation period for specific targets that are non-GPLed-only, not part of Community Edition.

From: John Mccabe <john@nospam.mccabe.org.uk>
Date: Mon, 29 Mar 2021 21:14:30 -0000

> The sales staff is pleasant to deal with,

That's good to know.

> but you might get sticker shock at the prices that they charge for non-GPLed supported products.

Possibly. It's been a long time since I knew the sort of prices these things go for, but it was in the thousands of dollars range then. It might still shock me though :-)

[...]

As far as evaluation goes, they do have a form that mentions it but it's the duration thing that would be an issue. I've tried to cultivate an interest in Ada amongst my colleagues (actually, my line manager's mostly done FPGA stuff using VHDL so some of the bits I've shown him have been 'familiar'), but we don't have anyone free to concentrate on evaluating something exclusively.

Targeting the 8051 with Ada

From: Mockturtle <framefritti@gmail.com>
Subject: Adapting an Ada compiler to generate 8051 code (Again?! ;-)
Date: Tue, 30 Mar 2021 02:04:41 -0700
Newsgroups: comp.lang.ada

for a project related to a possible start-up, we need to program a Flash controller that has a 8051 core (as many other controllers). I would like using Ada for that, but I discovered (also by browsing c.l.a.) that there is no Ada compiler producing 8051 code.

I am considering involving some university colleagues of mine to start a project aimed at having an Ada compiler for 8051, possibly leveraging some existing compiler. According to some posts read here, I understand that it is not totally impossible, if we are willing to accept some limitations.

I did not study (yet) in detail the 8051, but as I understand it is a small 8-bit processor, with flash memory for code and data and a small amount of RAM onboard (but maybe this depends on the specific controller). My knowledge about compilers is superficial, but I guess we should give up to some Ada features like

[List of runtime-based Ada features omitted. —arm]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 30 Mar 2021 11:56:34 +0200

I think the efforts would be better invested in recycling all existing 8051 cores. Make the planet greener! (:-))

Honestly, there is little useful one could do in 64K. Remember what one famous thinker and epidemiologist said about 640K? (640K is 10 times more than 64K)

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 30 Mar 2021 13:40:51 +0300

> [Original post. —arm]

I advise against that approach. The 8051 architecture is far too limited and quirky (and ancient) to waste such a major effort on.

However, you might have a look at the HAC compiler. As I understand it, it generates code for a virtual machine, and it might be easier to implement that virtual machine in 8051 code than it would be to generate 8051 code from the compiler.

[...]

I think you have two options:

1. Use HAC and implement the HAC VM in 8051 code, either in C or in assembler.

2. Pay for the AdaCore Ada-to-C compiler and use an 8051 C compiler as a back end.

[...]

There are some free 8051 C compilers (for example SDCC, Small Device C Compiler), but most professional programming for the 8051 uses commercial compilers such as the ARM/Keil compiler or the IAR compiler. You could try SDCC first, but if you get problems with e.g. using too much internal RAM, the commercial compilers might help.

I have often wished that there would be Ada compilers for more microcontrollers, but I understand why there aren't. An Ada-to-C compiler seems the most promising route.

From: Gautier <gautier_niouzes@hotmail.com>
Date: Tue, 30 Mar 2021 04:24:48 -0700

> Honestly, there is little useful one could do in 64K.

Well it depends...

On one hand there will never be enough memory (and cores) for the famous thinker's operating system just to run idle.

On the other hand you had some decades ago computers with everything stuffed in 64KB. For instance: a 16KB ROM with an OS, a BASIC interpreter, I/O, floating-point computations, etc.; 48KB RAM including the video memory. You had cool games and even a multi-window word processor on such a machine...

From: Mockturtle <framefritti@gmail.com>
Date: Tue, 30 Mar 2021 04:27:59 -0700

> Honestly, there is little useful one could do in 64K.

Well, the old ZX Spectrum with its 48K RAM extension (I and my brother said when we extended the RAM: "What are we going to do with all this memory?" :-D) used just 64K and you could do nice stuff. The first release of Turbo Pascal (editor and compiler integrated) was a .COM, limited by design to 64K.

I agree that it is easier to work without this limitation, but also the job of a flash microcontroller is not very complex.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 30 Mar 2021 14:01:34 +0200

> Well, old ZX Spectrum with its 48K RAM [...]

I remember the glorious time when 1K weighted 1kg (:-))

When I started, I and my pal worked together on a 256K machine in two time sharing terminal sessions. That was RSX-11M. These days almost every executable begins at 5-10M.

From: Paul Rubin
Date: Tue, 30 Mar 2021 12:16:46 -0700

> for a project related to a possible start-up, we need to program a Flash controller that has a 8051 core (as many other controllers).

Can you possibly avoid that? There are many microcontrollers that GCC has back ends for, so you could use GNAT. E.g. I think GNAT for the AVR is a thing. Of course even at the low end, ARM is everywhere now, and that is even easier.

Besides the approaches other people have mentioned, I don't know if there are any really large obstacles to targeting GCC to the 8051, or to some kind of VM that the 8051 can simulate, since you don't care about performance. If you do care about performance, you won't be using an 8051 in the first place ;-).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 31 Mar 2021 18:06:42 -0500

> Honestly, there is little useful one could do in 64K.

Gee, the early versions of Janus/Ada were *hosted* in 48K. Apparently, a compiler is nothing useful??? ;-)

We studied this problem back in the day (30+ years ago) The problem is the 8051 architecture, which doesn't have a usable stack or the instructions to make one. You would have to avoid recursion and any long chain of calls. Not sure whether the result would program much like Ada, it would be much closer to Fortran 66.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 31 Mar 2021 18:14:44 -0500

> I have often wished that there would be Ada compilers for more microcontrollers, but I understand why there aren't. An Ada-to-C compiler seems the most promising route.

Send $$$. ;-) This was a project that was ideally suited for the Janus/Ada compiler suite, but we never were able to find a customer for it. The problem is always that the first customer has to pay a substantial part of the development; later customers don't have to pay that freight. (Back in the "waiver" days we considered doing it for the "fun" of making DoD-types have to find better excuses to avoid Ada than a compiler not existing for it, but the likely ROI wasn't there to convince the angel investors to go along with the idea.)