AdaCore
The GNAT Pro Company

Язык Ада в современной программной индустрии.

Сергей Рыбин,

rybin@adacore.com

Октябрь 2018, Орел

О чем пойдет речь...

- **Язык Ада: современное состояние и использование в программной индустрии.**
- **Система программирования GNAT и компания AdaCore.**
- **Технологические решения на основе Ады.**
- **Вопросы?**

Ада – прошлое, настоящее, будущее...

- **«Золотой век» Ады:** конец 70-х – середина 90-х годов прошлого века.

- **Уверенно себя чувствует в своей нише**:
  - Авиация (встроенное ПО и системы управления воздушным движением);
  - Финансы;
  - Транспорт;
  - Связь и телекоммуникации;
  - Атомная энергетика;
  - Космос;
  - Медицина;
  - ...

- **Тенденция к расширению ниши**

Ада – кто сейчас ее использует

- Alenia
- Alstom Transport
- Ansaldo STS
- Atos Origin
- AWE
- BAE Systems
- Boeing
- EADS
- European Space Agency
- Eurocontrol
- IPESOFT
- JEOL
- Lockheed Martin
- MBDA
- Philips Semiconductor
- Raytheon
- Rockwell Collins
- SAAB
- General Electric
- Thales

Ада – стандартизация

- **Ада возникла как стандарт ANSI (1983), утверженный как стандарт ISO (1987).**

- **Правила ISO требуют пересматривать информационный стандарты раз в 10 лет, Ада – единственный язык, следующий этому правилу**: 1987 => 1995 => 2005 => 2012 (последняя официальная ревизия стандарта) => 202X (в процессе разработки)

- **Опережающая стандартизация +мощные средства контроля реализаций = отсутствие версий и диалектов языка.**

Ада и Оберон – что у них общего?

- **Ада и Оберон – близкие родственники, у них общий дедушка (Виртовский Паскаль).**

- **Общая философия:**
  - Забота о надежности на всех этапах жизненного цикла ПО:
    - Строгая типизация, отсутствие умолчаний, «все, что не разрешено – запрещено»;
  - Сопоставимый набор предоставляемых возможностей;
  - Похожий (понятный и легко читаемый!) синтаксис;

Ада и Оберон – в чем разница?

Принцип сундука и принцип чемоданчика

(Кауфман В.Ш. «Языки программирования. Концепции и принципы» ДМК Пресс, 2011)

- **«Принцип чемоданчика»:** небольшой набор абсолютно необходимых инструментов, все остальное сделаем сами по месту => **Оберон**

- **«Принцип сундука»:** для каждой базовой (да и вообще важной) технологической потребности должно быть готовое решение или легко настраиваемый полуфабрикат => **Ада**

Что есть интересного в Адском сундуке?

- **Управление асинхронными процессами;**
- **Иерархическая модульность;**
- **Механизм подтипов (вплоть до определения произвольных множеств) и механизм исключений;**

```
subtype Basic_Letter is Character
   with Static_Predicate => Basic_Letter in 'A'..'Z' | 'a'..'z' | 'Æ'
                                          | 'æ' | 'Ð' | 'ð' | 'Þ' | 'þ' | 'ß';

subtype Even_Integer is Integer
   with Dynamic_Predicate => Even_Integer mod 2 = 0,
        Predicate_Failure =>
          "Even_Integer must be a multiple of 2";
```

- **Атрибуты как средство запросить свойства сущностей и аспекты как средство задать свойства сущностей (от размера переменной до пред- и пост-условий для подпрограммы)**

Что есть интересного в Адском сундуке?

- **Исполняемые спецификации (предикаты, пред- и пост-условия, инварианты);**
- **Кванторы всеобщности и существования в логическом выражении;**

```
-- постусловие для подпрограммы, параметром которой является
-- тип-массив A с типом индекса T (требование упорядоченности
-- результата):

Post => (A'Length < 2 or else
   (for all I in A'First .. T'Pred(A'Last) => A (I) <= A (T'Succ (I))))

pragma Assert (for some X in 2 .. N / 2 => N mod X = 0);
```

- **Возможность определять технологическое подмножество языка;**

```
pragma Restrictions (No_Tasking);

pragma Profile (Ravenscar);
```

- **И много еще всякого-разного ...**

Ада и разработка встроенных приложений

Ада изначально ориентирована на разработку встроенных систем:

- Все, что можно, сдвинуто на этап компиляции и сборки программы;
- Спецификации представления – управление машинно-зависимыми аспектами (задание адресов объектов, размера значений, расположения составных данных в памяти и т.п.);
- Интерфейс с другими языками;

И все это – не выходя за рамки стандарта языка!

Ада - реализации

- GNAT – AdaCore;
- GNAT – FSF;
- GNAT – от всех, кому не лень;
- Старые игроки (Rational, Aonix, DDC-I, Green Hills) все еще в деле, но...

Компания AdaCore (www.adacore.com)

- Образована в 1994 году преподавателями Нью-Йоркского университета;
- В настоящее время – более 100 сотрудников, головные офисы в Париже и Нью-Йорке;
- Занимается разработкой и сопровождением GNAT-технологии;
- Основной продукт компании – подписка на оказание технической поддержки для пользователей GNAT-технологии;

GNAT = Gnu Nyu Ada Translator

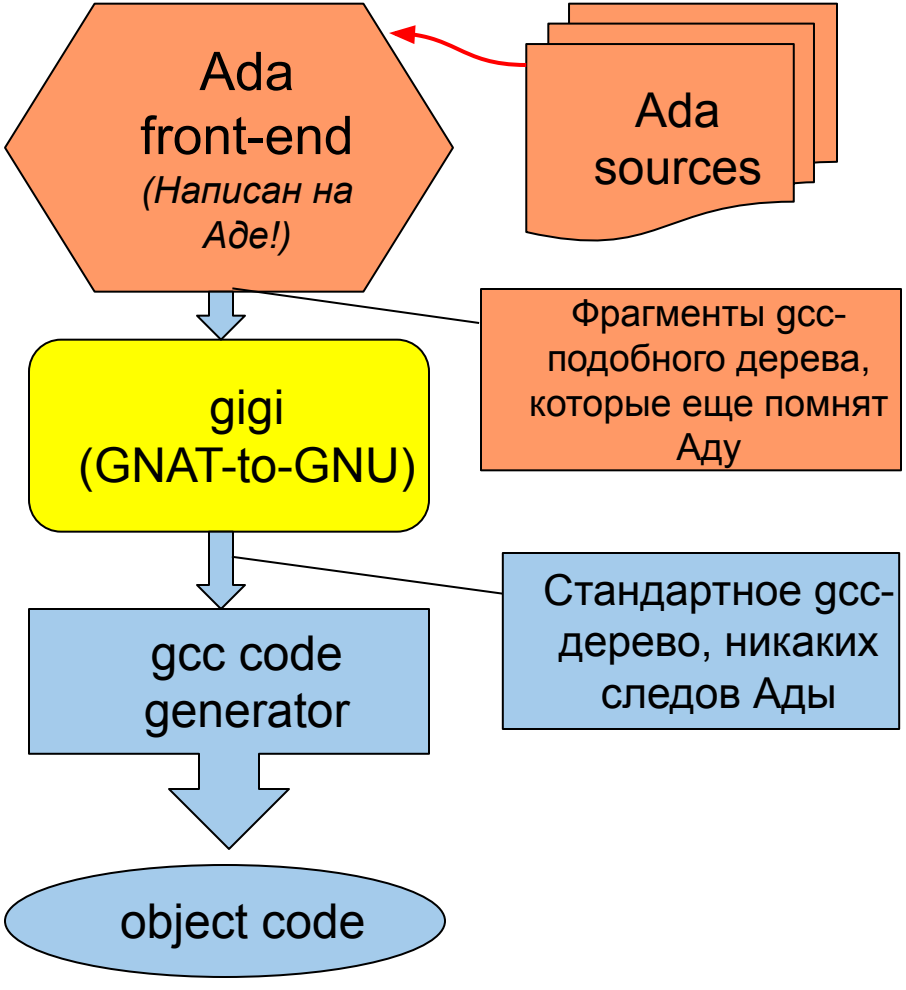
Доступен в инструментальных средах:

- **AIX**
  PowerPC AIX (32 bits)
- **HP-UX**
  Itanium HP-UX
- **Linux**
  HP Integrity Itanium GNU Linux
  PowerPC GNU Linux (32 bits)
  SGI Altix Itanium GNU Linux
  x86 GNU Linux (32 bits)
  x86-64 GNU Linux (64 bits)
- **Mac OS X**
  x86-64 Mac OS X (64 bits)
- **Solaris**
  SPARC Solaris (32 bits)
  SPARC Solaris (64 bits)
  x86 Solaris/Trusted Solaris (32 bits)
- **Windows**
  x86 Windows (32 bits)
  x86-64 Windows (64 bits)

GNAT – позволяет создавать код для:

- **Android**
  - ARM Android (hosted on Linux)
  - ARM Android (hosted on Windows)
- **Certified VxWorks**
  - PowerPC VxWorks 6.x/Cert (hosted on Linux)
  - PowerPC VxWorks 6.x/Cert (hosted on Windows)
  - PowerPC VxWorks 653 (hosted on Windows)
  - PowerPC VxWorks MILS (hosted on Windows)
- **ELF format**
  - ARM ELF format (hosted on Linux)
  - ARM ELF format (hosted on Windows)
  - ERC32 ELF format (hosted on Linux)
  - ERC32 ELF format (hosted on Solaris)
  - LEON 2 ELF format (hosted on Linux)
  - LEON 2 ELF format (hosted on Solaris)
  - LEON 2 ELF format (hosted on Windows)
  - LEON 3 ELF format (hosted on Linux)
  - LEON 3 ELF format (hosted on Windows)
  - PowerPC 55xx and e500v2 ELF format (hosted on Linux)
  - PowerPC 55xx and e500v2 ELF format (hosted on Windows)
  - PowerPC ELF format (hosted on Solaris)
  - PowerPC ELF format (hosted on Windows)
- **Embedded Linux**
  - ARM GNU Linux (32 bits) (hosted on Linux)
  - PowerPC ELinOS (hosted on Linux)
  - x86 ELinOS (hosted on Linux)
- **Linux**
  - PowerPC e500v2 GNU Linux (32 bits) (hosted on Linux)
- **LynxOS**
  - PowerPC LynxOS 4.x (hosted on Solaris)
- **PikeOS**
  - x86 PikeOS (hosted on Linux)
- **VxWorks 5.x**
  - PowerPC VxWorks 5.x (hosted on Solaris)
  - PowerPC VxWorks 5.x (hosted on Windows)
- **VxWorks 6.x**
  - ARM VxWorks 6.x (hosted on Linux)
  - ARM VxWorks 6.x (hosted on Windows)
  - PowerPC VxWorks 6.x (hosted on Linux)
  - PowerPC VxWorks 6.x (hosted on Solaris)
  - PowerPC VxWorks 6.x (hosted on Windows)
  - PowerPC e500v2 VxWorks 6.x (hosted on Linux)
  - PowerPC e500v2 VxWorks 6.x (hosted on Solaris)
  - PowerPC e500v2 VxWorks 6.x (hosted on Windows)
  - x86 VxWorks 6.x (hosted on Linux)
  - x86 VxWorks 6.x (hosted on Solaris)
  - x86 VxWorks 6.x (hosted on Windows)
- **VxWorks 7.x**
  - ARM VxWorks 7.x (hosted on Linux)
  - ARM VxWorks 7.x (hosted on Windows)
  - PowerPC VxWorks 7.x (hosted on Linux)
  - PowerPC VxWorks 7.x (hosted on Windows)
  - PowerPC e500v2 VxWorks 7.x (hosted on Linux)
  - PowerPC e500v2 VxWorks 7.x (hosted on Windows)
  - x86-64 VxWorks 7.x (64 bits) (hosted on Linux)
  - x86-64 VxWorks 7.x (64 bits) (hosted on Windows)
- **Wind River Linux**
  - PowerPC Wind River Linux (hosted on Linux)
  - PowerPC e500v2 Wind River Linux (hosted on Linux)

Компилятор GNAT



- **Преимущества использования gcc:**
  - Ада есть (может быть) везде, где есть gcc;
  - Многоязыковое программирование (Ада может использоваться совместно с любым языком, реализованным в gcc);
  - GNAT – среда программирования для Ada/C/C++

- **GNAT way of using gcc:**
  - Параноидально-консервативный: только стабильные версии, только надежные возможности;
  - Набор своих изменений для gcc;

GNAT-технология

Не только компилятор Ады (поддерживается последняя версия стандарта), но и:

- **Компиляторы C и C++;**
- **Специализированные библиотеки;**
- **Инструментарий;**
- **Технологические решения;**
- **SPARK (доказательное программирование);**
- **QGen (автоматическая генерация кода по моделям);**
- **…**

AdaCore - продукты

- **GNAT Pro:**
  - Полноценная техническая поддержка;
  - Возможность создавать программы с закрытым кодом;
  - Все платформы и все целевые среды, весь инструментарий, все компоненты технологии;

- **GNAT Developer:**
  - Незначительно ограниченная техническая поддержка;
  - Возможность создавать программы с закрытым кодом;
  - Незначительные ограничения на поддерживаемые конфигурации;

- **GAP (GNAT Academic Program):**
  - Ограниченная техническая поддержка;
  - Возможность создавать программы только под лицензией GPL;
  - Все инструментальные платформы + MINDSTORMS NTX robotic + базовый инструментарий;
  - Бесплатно для университетов;

- **GNAT GPL:**
  - То же, что и GAP, но вообще без техподдержки и бесплатно абсолютно для всех;

AdaCore – технологические решения

- **Разработка и тестирование встроенных приложений;**
- **Поддержка разработки сертифицированных приложений;**
- **Автоматическая генерация Ада-кода на основе моделей;**
- **Формальная верификация программ, или корректность по построению;**
- **Обучение, консультации, развитие технологии на основе явных заказов пользователей;**

Разработка и тестирование встроенных приложений

- **Конфигурируемые библиотеки периода выполнения:**
  - Ravenscar profile;
  - Zero footprint;
  - Bare board (возможность создавать приложения при отсутствии ОС в целевой среде);

- **gnat emulator –** возможность тестировать встроенное приложение не в целевой, а в инструментальной среде;

Разработка сертифицированных приложений

Поддержка разработки систем, удовлетворяющих отраслевым стандартам качества:

- DO-178 (встроенные авиационные системы);
- EN 50128 (железные дороги);
- RTCA DO-278 (управление воздушным движением);
- ECSS-E-ST-40C, ECSS-Q-ST-80C – космическая техника;
- ...

Разработка сертифицированных приложений

Поддержка тестирования:

- **gnatcoverage**: автоматическая проверка выполнения критериев структурного тестирования (как на уровне объектного кода, так и на уровне исходного кода на языках Ада и C);
- **gnattest**: автоматическая генерация тестовых драйверов для выполнения модульного тестирования в среде aunit;

Traceability analysis package:

- рекомендации по выбору языкового подмножества, позволяющего проследить соответствие исходного и объектного кода, и нужных режимов компилятора;
- набор тестов, демонстрирующих соответствие исходного и объектного кода в пределах выбранного подмножества;

Разработка сертифицированных приложений

Инструменты статического анализа кода:

- **gnatcheck:**
  - Проверка правил, типичных для стандарта кодирования;
  - Интеграция результатов проверок, проводимых компилятором, в итоговый отчет;
  - Позволяет быстро добавлять новые правила по заказу пользователей;
- **Codepeer**:
  - Детальный анализ графа управления и графа потока данных (деление на ноль, переполнение, неопределенные и неиспользуемые переменные и т.п.);
- **gnatstack:**
  - Оценка максимальной глубины стека в целевой среде;
  - Выявление потенциально опасных ситуаций;
  - Работает с кодом на Аде, C, C++.

Разработка сертифицированных приложений

- **Сертифицированные библиотеки периода выполнения;**
- **Предоставление квалификационных материалов для (части) предлагаемых инструментов и решений;**

Автоматическая генерация Ада-кода на основе моделей

QGen:

- **Поддерживается надежное подмножество Simulink® and Stateflow®;**
- **Проверка корректности модели;**
- **Возможность проследить путь от модели к коду;**
- **Генерация кода на SPARK и MISRA C;**
- **Цель – полностью квалифицированная технология для использования там, где действуют стандарты безопасности;**

SPARK – формальная верификация программ

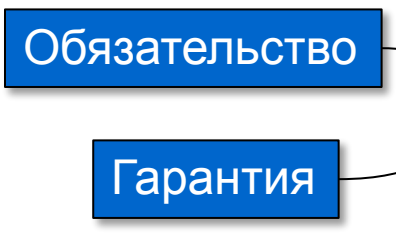
- **С чего начался SPARK:**
  - (Успешная!) попытка практической реализации идеи формального доказательства корректности программы;
  - Использование булевских аннотаций для компонент кода на Аде в виде комментариев специального вида (пред- и пост-условия для подпрограмм, инварианты циклов, произвольные утверждения);
  - Крайне ограниченное подмножество Ады;
  - Бескомпромиссный подход «всё или ничего»;
  - Несколько успешных индустриальных проектов;

- **Развитие SPARK-технологии:**
  - Аннотации, инварианты, утверждения стали исполняемыми компонентами кода на Аде;
  - Новые способы аннотации;
  - Существенно расширилось подмножество Ады;
  - Более гибкий подход – комбинация:
    - Формального доказательства корректности программных модулей (возможно, не всех);
    - тестирования;
    - обнаружения потенциальных проблем (или доказательства их отсутствия!);

  Как результат – намного более широкое использование в индустрии;

Контракт как ключевая идея SPARK-подхода

- **Определяемые пользователем проверки, осуществляемые при выполнении программы;**
- ***Предусловия* – обязательства того, кто вызывает подпрограмму;**
- ***Постусловия* – гарантии того, что произойдет в результате выполнения подпрограммы**




```
procedure Push (This : in out Stack;  Value : Content) with
      Pre  =>   not Full (This),
      Post =>   not Empty (This) and Top (This) = Value;
…
function Top (This : Stack) return Content;
function Full (This : Stack) return Boolean;
```

Почему это назвали контрактом?

- **Если вызывающий контекст гарантирует, что предусловие вызываемой подпрограммы истинно, то подпрограмма обеспечивает истинность пост-условия по завершению вызова;**

- **Все пред- и пост-условия вычисляются как часть вызова подпрограммы, и если что-либо не оказывается истинным – возбуждается исключение;**

КОНТРАКТ	Пред	Пост
Пишущий код, где происходит вызов	Обязан обеспечить истинность	Предполагает истинность по завершении вызова
Пишущий тело вызываемой подпрограммы	Предполагает истинность перед вызовом	Обязан обеспечить истинность по завершении вызова

Инварианты типов (вишенка на торте :)

```
package Stacks is
   type Stack is private
      with
         Type_Invariant => Is_Unduplicated (Statck);

   function Is_Empty (S : Stack) return Boolean;
   function Is_Full (S : Stack) return Boolean;
   function Is_Unduplicated (S : Stack) return Boolean;

   procedure Push (S : in out Stack; X : Integer);
      with
         Pre  => not Is_Full (S);
         Post => Is_Empty (S);
   ...
end Statcks;
...
Var : Stack;
...

Push (Var, 13);
--  Is_Unduplicated будет вызвана сразу же после завершения этого вызова,
--  и если результатом будет FALSE, будет возбуждено исключение
```

Инвариант типа:

- Проверяется для любого объекта типа после любого действия, способного изменить значение объекта;
- Наследуется для типов-потомков;
- Может быть переопределен для типа-потомка;