H A C

HAC Ada Compiler

HAC Tutorial

G. de Montmollin, Ada-Europe 2023

HAC Tutorial

Plan

1. Presentation of HAC, the HAC Ada Compiler – **09:00**
2. Overview of HAC's Ada subset, basic types – **09:30**
3. Range checks, exceptions – **10:00**

– Break: **10:30-11:00** –

4. Packages – **11:00**
5. Compatibility with "full Ada" – **11:30**
6. Virtual Machine, customization, integration of HAC into another software – **12:00**

1. Presentation **HAC**'s history

1990's: Was looking for a small, quick Ada compiler. Found SmallAda, an abandoned but operational project, including a programmable type system (arrays, records), nested subprograms, tasks, recursion, etc., compiling towards a VM. Origin: Pascal-S, then Co-Pascal

1999: Unsuccessful attempt to translate SmallAda from Pascal to Ada, using **P2Ada**.

2009: Improved **P2Ada** to translate the full SmallAda sources.

2013: January 24th: **Day Zero** of **HAC**. Hello World, Fibonacci, sorting demos and few other tests work!

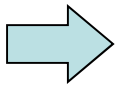
2020: After a long sleep, the real development of **HAC** begins!

1. Presentation **HAC**'s goals

- **General motivation**: provide a simple, easy, quick compiler
  - users: beginners, non IT specialists, or even IT specialists
  - focus: teaching, small programs such as text parsers, file converters, shell scripts launching applications, code generators, numerical simulations, …

- **Former goal**: make it work. Have an Ada compiler that produces "Hello world!" with a tiny object code size.

```
with HAT;
procedure Hello is
begin
  HAT.Put ("Hello world!");
end Hello;
```



```
Position: Opcode                          Approx source location
-----------------------------------------------
       0: K_PUSH_TWO_DISCRETE_LITERALS     4  Hello
       1: K_PUSH_TWO_DISCRETE_LITERALS     4  Hello
       2: K_FILE_I_O                       4  Hello; SP_PUT; "Hello world!"
       3: K_HALT_INTERPRETER               5  [-- The Universe --]
```

1. Presentation **HAC**'s goals

- **New goal**: break the language wall between the "script" world and the "compiled" world ("two-language problem")

Prototyping, data processing, data analysis, code generation, simulations, shell scripts, plug-ins, exercises, …

Large applications

Production code

Domain specific

e.g. SQL

Traditionally: dynamic, duck typing; interactive workflow with global, persistent data

Static typing (for performance), data loaded and managed by the application

HAC ← **Ada** → "full Ada" system

1. Presentation **HAC**'s 2020+ development

- Made the parser Ada compatible
- Exception trace-backs
- HAT (HAC Ada Toolbox) package with lots of goodies
- Variable-length strings (type HAT.VString)
- Text files (type HAT.File_Type)
- Subtypes
- Range checks
- Modularity (library-level packages, procedures and functions)
- 

Interfacing HAC ↔ native "full Ada"

Behind the scenes:

- Regression test suite with 72 tests (55 thanks to Advent of Code)
- Compiler: global variables replaced by OO
- Parser, VM interpreter were split into child packages, then enhanced

1. Presentation **HAC**'s 2020+ development

Behind the scenes (continued):

Long time ago (Pascal-S, 1973):

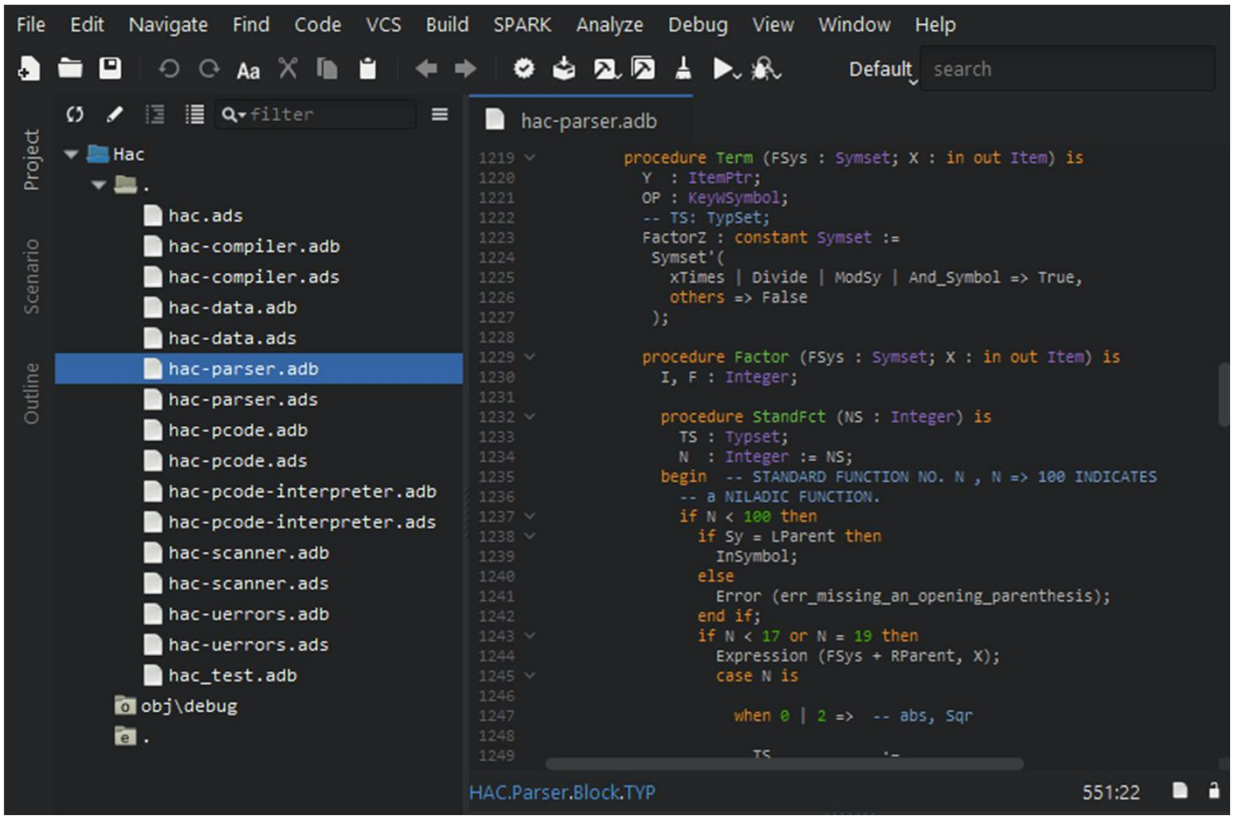

```
          var y:item; op:symbol;

          procedure term(fsys:symset; var x:item);
            var y:item; op:symbol; ts:typset;

            procedure factor(fsys:symset; var x:item);
              var i,f: integer;

              procedure standfct(n: integer);
                 var ts: typset;
              begin (*standard function no. n*)
                if sy = lparent then insymbol else error(9);
                if n < 17 then
                  begin expression(fsys+[rparent],x);
                    case n of
(*abs,sqr*)          0,2:  begin ts := [ints,reals];
                             tab[i].typ := x.typ;
                             if x.typ = reals then n := n+1
                           end ;
(*odd,chr*)          4,5:  ts := [ints];
(*ord*)              6:    ts := [ints,bools,chars];
(*succ,pred*)        7,8:  ts := [chars];
(*round,trunc*)      9,10,11,12,13,14,15,16:
```

1. Presentation **HAC**'s 2020+ development

Behind the scenes (continued):

Around 2014: still mostly Pascal-S!

- Many globals
- Many magic numbers
- Many variables (global and local) with the same cryptic name (I, F, N, …)



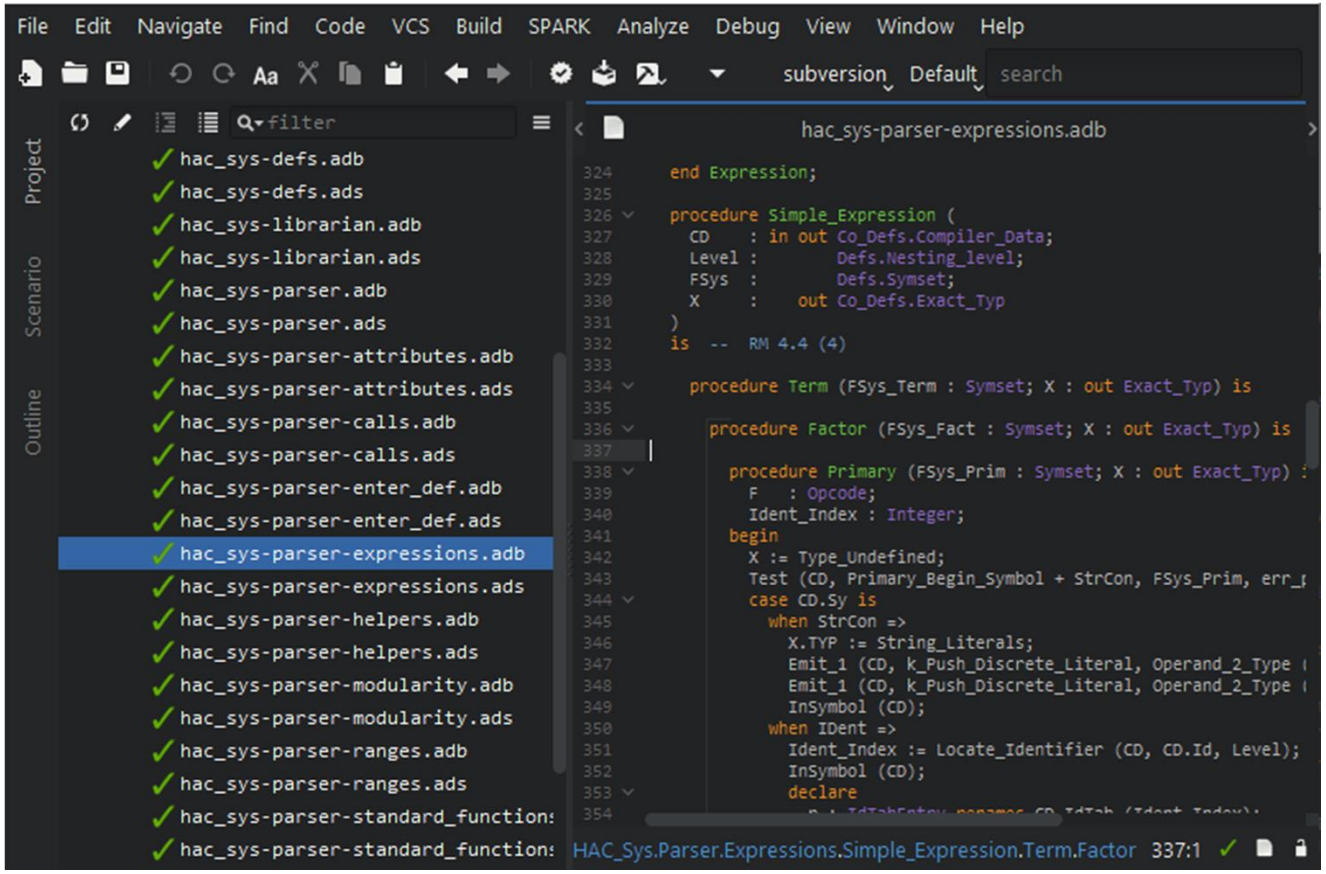
1. Presentation **HAC**'s 2020+ development

Behind the scenes (continued):

Now:

-Parser and VM split into child packages

-Object-oriented



1. Presentation — **HAC**'s characteristics

- **Build time** of the full HAC compiler & VM interpreter, by GNAT: 5.6 seconds (i7-9700 CPU @ 3.00GHz, using 8 cores)
- **Build time** of exm/sudoku_sample.adb (3 units, 1355 lines):
  - by HAC: 0.09 second (single task)
  - by GNAT: 2.4 seconds (multi-core, without recompiling HAT)
- **System dependency**: none
- **Ada source input**: any stream (file, editor data (e.g. LEA), web stream, zip archive, …)
- **Target**: Virtual Machine (p-code with many extensions)

- **More Ada features**

- **Built-in libraries:**
  - SQLite
  - Some portable graphical toolkit
  - Ada.*
  - More of Interfaces.*
  - UTF-8 functions

1. Presentation — Where to find **HAC** ?

HAC is free, open-source (MIT license)

- **SourceForge:**
  - **Home page:** https://hacadacompiler.sourceforge.io/
  - **Project page:** https://sourceforge.net/projects/hacadacompiler/

- **Github:** https://github.com/zertovitch/hac

1. Presentation — Projects related to **HAC**

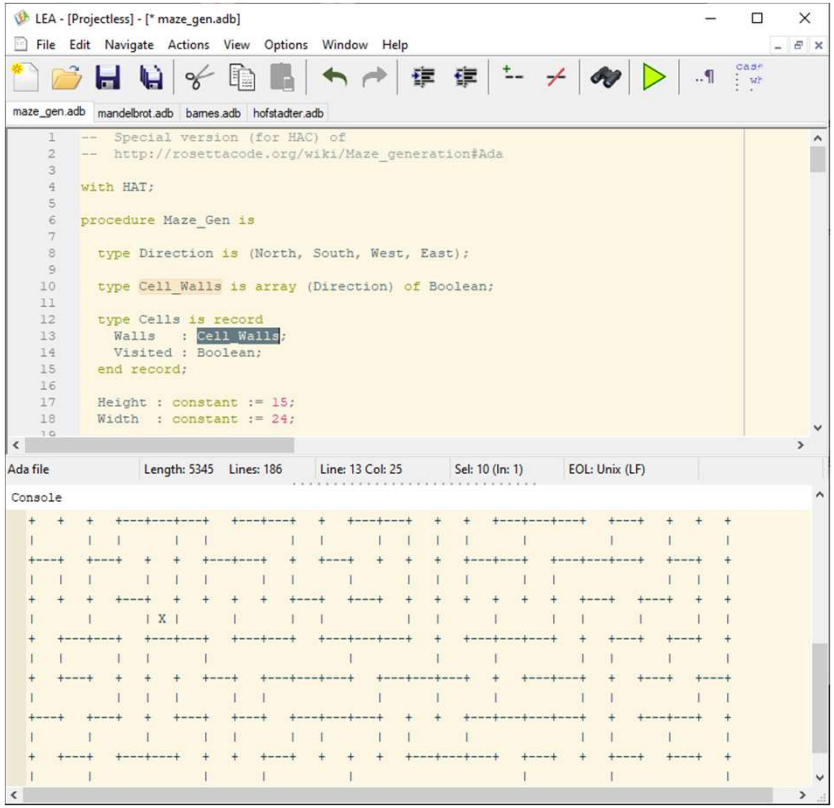
LEA (**L**ightweight **E**ditor for **A**da)

- https://sourceforge.net/projects/l-e-a/
- https://github.com/zertovitch/lea

LEA embeds HAC (through Ada's modularity).

NB:

- HAC embeds its own VM and run-time library, including the body of HAT package
- lea.exe is a single-executable application; doesn't need installation



Pascal-to-Ada (was needed for the genesis of HAC)

- https://sourceforge.net/projects/p2ada/
- https://github.com/zertovitch/pascal-to-ada

2.1 **HAC**'s Ada subset (v.0.2): basic types

- **Boolean**
- **Integer** (signed 64 bit)
  - Natural
  - Positive
- ~~**Float**~~ → HAT.Real (IEEE double precision or more)
- **Character**
- **String**… implemented in a limited way: literals, results of attributes, constrained (String (a .. b)) → you are invited to use HAT.VString for practical strings
- **Duration**

2.2 **HAC**'s Ada subset (v.0.2): user-defined types

- **Arrays (constrained)**
- **Records** (without parameter or tag)
- Unlimited combinations of the above ☺
- No **access types**

2.3 **HAC**'s Ada subset (v.0.2): declarations

- User-defined **types, subtypes**
- **Objects** (constants & variables)
- **Numbers**
- **Subprograms** (incl. null procedures) (*)
- **Packages** (*)
- **Tasks**: partially successful revival of tasking of SmallAda, the ancestor of HAC 

- Not yet: **renames, user-defined exceptions, generics**

(*) Local to a subprogram, or as library-level unit

2.4 **HAC**'s built-in library (HAT)

HAT: the **HAC A**da **T**oolbox

- **Includes basic types and functions**
- **VString** (variable-length strings, renaming of Unbounded_String)
- **Text I/O**
- **Ada.Command_Line**-like
- **Ada.Directories**-like
- **Ada.Environment_Variables**-like
- Timing (**Ada.Calendar**-like)
- Basic math functions (**Ada.Numerics.Elementary_Functions**-like)
- **Shell_Execute** (calls command-line interpreter with a command)

3.1 **HAC**'s built-in checks

Compile-time range checks

```
subtype A is Integer range 1 .. 100;
subtype B is A range 1 .. 1000;
```

```
Error in range constraint: higher bound, 1000, is out of parent type's
range, 1 .. 100
```

Other compilers prefer to emit a warning, e.g.

```
warning: "Constraint_Error" will be raised at run time
```

3.2 **HAC**'s built-in checks

Run-time *and* compile-time range checks

- Array index within array's range
- Assignment (`x := expression`)
- Type conversions
- `S'Pred, S'Succ` attributes

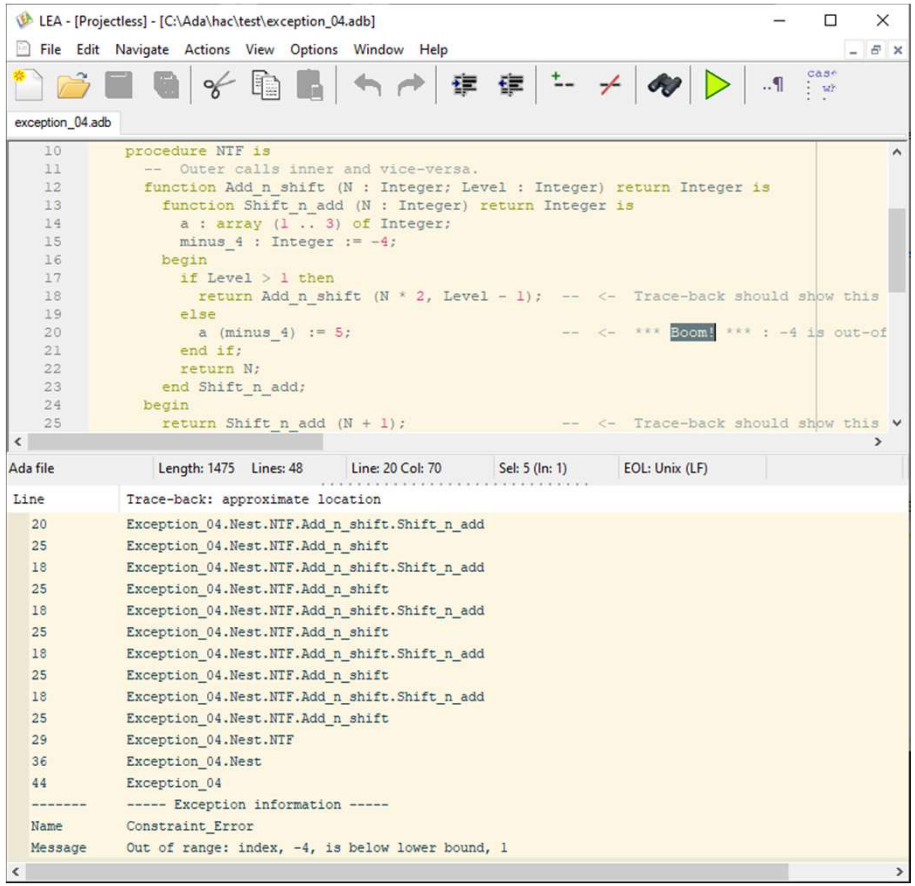
3.3 **HAC**'s built-in checks

Other run-time checks

- Function's "`end`" reached before a "`return`" (`Program_Error`)
- Stack overflow (`Storage_Error`)
- I/O errors: `Name_Error`, `Use_Error`, `End_Error` (they are just propagated from the Ada host system since the Virtual Machine is programmed in Ada ☺)

3.4 **HAC**'s exceptions

- No **user-defined exceptions** yet
- No **handlers** yet
- But… **HAC** provides
  trace-backs
  by default:



4.1 **HAC**'s packages: built-in

- **Standard** (parts)
- **Interfaces** (parts)
- **HAT** (with VStrings, Text I/O, Command line, basic numerics, …). Was "`with & use`"-ed by default in early HAC versions (and in SmallAda, CoPascal, etc.). Now semantically a normal package.

- Planned: database package: **SQLite**
- Planned: <your input> !

4.2 **HAC**'s packages: user-defined

User-defined packages are recent in HAC: were added in April 2022.

- Local packages (package as local definition in a subprogram) ✓

- Library-level packages (modules):

  Limitation: nothing requiring initialization code or room on the stack → no constant, no variable, no initialization part

4.3 **HAC**: library-level packages

- A naming convention is needed, since HAC discovers automatically the library during the build phase (like GNAT or Borland Pascal / Delphi).

- Naming convention is GNAT default's:

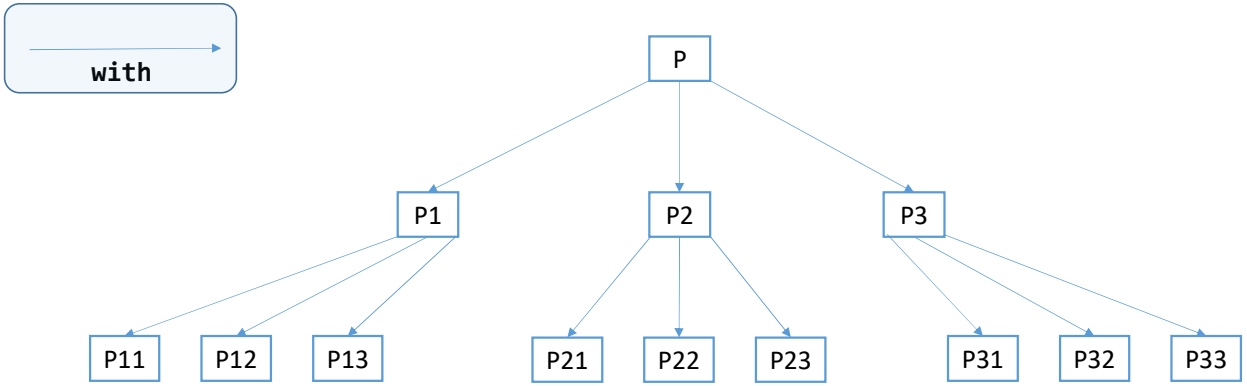
  parent-child.ad**s** : specification

  parent-child.ad**b** : body

4.4 **HAC**: library-level packages

Automatic library discovery: when parsing "`with X`", HAC looks for **X** in the library.

- If already there, X's declarations are reactivated.
- If not, HAC suspends the current compilation and X is compiled (unit specification or subprogram proper body). In case of a specification, compilation of body is queued.



Variants: «`with`» in specification, body, or mixed (random choice)

4.5 **HAC**: main procedure

For main procedure, file name extension's choice is free (possible since file name is given to HAC):

- tool.adb
- tool.hac

Trick for a file explorer: associate ".hac" files with the HAC command-line tool (Linux: hac, Windows: hac.exe)

5.1 **HAC**: Compatibility with “full Ada”

HAC covers a (currently growing) subset of the Ada language. Most of the time the compiler checks correctly the of Ada compliance.

Known semantics issues (v.0.2)

- HAT’s operators (“+”, “-”, “&”) always visible
- Array indexing: can write (x,y) or (x)(y) (Pascal semantics)

Solution: compile from time to time with another compiler.

NB: this remark is also valid for compilers other than HAC…

5.2 **HAC**: Compatibility with “full Ada”

In practice:

- Shipped with HAC:
  - **examples**: ~110, of which 66 Advent of Code (AoC) programs
  - **tests**: 78. Most of the AoC programs are also part of HAC’s automated regression test.
  - GNAT & ObjectAda project files for „full Ada“ runs of the above

- “Pro” usage:
  - code generators
  - off version-control backups
  - dynamic configuration (HAC embedded within an application, cf. 6.6)
  - large server configuration programs (developed with HAC, deployed with HAC or GNAT)

6.1 **HAC**: The Virtual Machine

Origin: Pascal-S p-code

```
type Opcode is
 (k_Push_Address,
  k_Push_Value,
  k_Push_Indirect_Value,
  k_Push_Discrete_Literal,
  k_Push_Float_Literal,
  …
```

6.2 **HAC**: The Virtual Machine

Memory:

- A stack (partitioned for tasks)
- No heap
- No fixed registers until recently (1 temporary register was added for a special purpose)

A memory cell:

```
type General_Register (Special : Defs.Typen := Defs.NOTYP) is record
  -- I is used for most uses: indices in the stack, Integers, Bools, Chars and Enums.
  I : Defs.HAC_Integer;
  case Special is
  -- This part is variant to save place.
    when Defs.Floats     => R   : Defs.HAC_Float;
    when Defs.VStrings   => V   : HAT.VString;
    when Defs.Times      => Tim : Ada.Calendar.Time;
    when Defs.Durations  => Dur : Duration;
    when Defs.Text_Files => Txt : File_Ptr := Abstract_Console;
    when others          => null;
  end case;
end record;
```

6.3 **HAC**: The Virtual Machine

VM Interpreter – excerpt 1: basic virtual processor operations

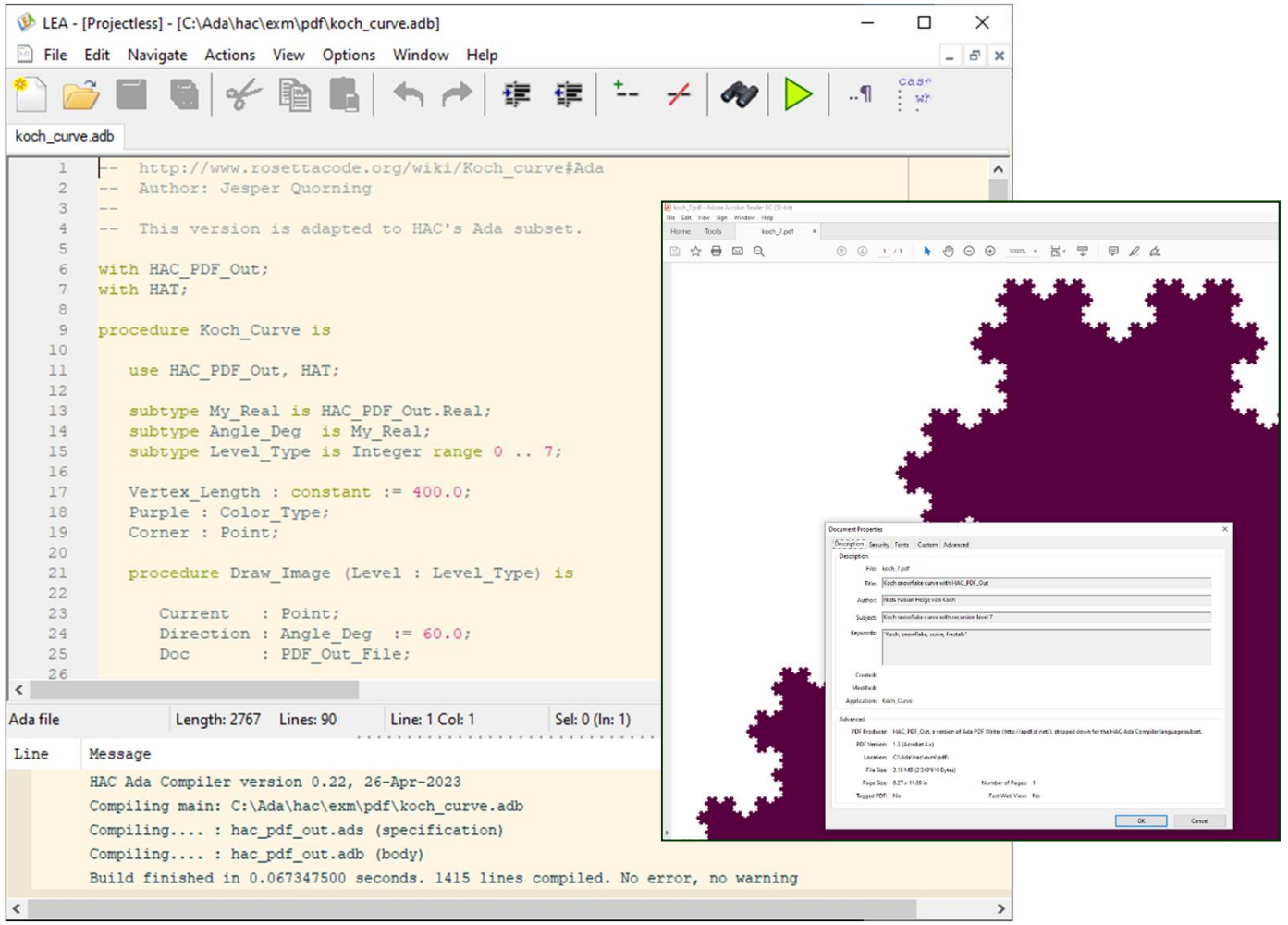
```
procedure Do_Atomic_Data_Push_Operation is
  Base, Address_of_Variable : Index;
begin
  Push;
  if ND.IR.F in k_Push_Address .. k_Push_Indirect_Value then
    Base                := Curr_TCB.DISPLAY (Nesting_level (IR.X));
    Address_of_Variable := Base + Index (IR.Y);
  end if;
  case Atomic_Data_Push_Opcode (ND.IR.F) is
    when k_Push_Address =>
      --  Push "v'Access" of variable v
      ND.S (Curr_TCB.T).I := HAC_Integer (Address_of_Variable);
    when k_Push_Value =>
      --  Push variable v's value.
      ND.S (Curr_TCB.T) := ND.S (Address_of_Variable);
    when k_Push_Indirect_Value =>
      --  Push "v.all" (variable v contains an access).
      ND.S (Curr_TCB.T) := ND.S (Index (ND.S (Address_of_Variable).I));
    when k_Push_Discrete_Literal =>
      --  Literal: discrete value (Integer, Character, Boolean, Enum)
      ND.S (Curr_TCB.T).I := IR.Y;
    when k_Push_Float_Literal =>
      ND.S (Curr_TCB.T) := GR_Real (CD.Float_Constants_Table (Integer (IR.Y)));
    when k_Push_Float_First =>
      ND.S (Curr_TCB.T) := GR_Real (HAC_Float'First);
    when k_Push_Float_Last =>
      ND.S (Curr_TCB.T) := GR_Real (HAC_Float'Last);
  end case;
end Do_Atomic_Data_Push_Operation;
```

6.4 **HAC**: The Virtual Machine

VM Interpreter – excerpt 2: HAT (gives ideas for customization)

```
begin
  case Code is
    when SP_Set_Env         => HAT.Set_Env    (Below_Top_Item.V, Top_Item.V);
    when SP_Set_VM_Variable =>
      Interfacing.Set_VM_Variable
        (BD, To_String (Below_Top_Item.V), To_String (Top_Item.V));
    when SP_Copy_File       => HAT.Copy_File (Below_Top_Item.V, Top_Item.V);
    when SP_Rename          => HAT.Rename     (Below_Top_Item.V, Top_Item.V);
    when SP_Delete_File     => HAT.Delete_File (Top_Item.V);
    when SP_Set_Directory   => HAT.Set_Directory (Top_Item.V);
    when SP_Set_Exit_Status => HAT.Set_Exit_Status (Integer (Top_Item.I));
```

6.5 **HAC**: Integration example: LEA



6.6 **HAC**: Integration into other software

Minimal standalone application

```
with Ada.Command_Line, Ada.Text_IO;
with HAC_Sys.Builder, HAC_Sys.PCode.Interpreter;

procedure HAC_Mini is
  use Ada.Command_Line, Ada.Text_IO;
  use HAC_Sys.PCode.Interpreter;
  --
  BD : HAC_Sys.Builder.Build_Data;
  post_mortem : Post_Mortem_Data;
begin
  if Argument_Count = 0 then
    Put_Line (Current_Error, "Usage: hac_mini main.adb");
  else
    BD.Build_Main_from_File (Argument (1));
    if BD.Build_Successful then
      Interpret_on_Current_IO (BD, 1, "", post_mortem);
      if Is_Exception_Raised (post_mortem.Unhandled) then
        Put_Line (Current_Error, "HAC VM: raised " & Image (post_mortem.Unhandled));
        Put_Line (Current_Error, Message (post_mortem.Unhandled));
      end if;
    end if;
  end if;
end HAC_Mini
```

6.6 **HAC**: Integration into other software

Communication between the HAC program and its hosting program

- Your native or embedded Ada program can compile and run itself Ada programs using HAC semantics subset, while running → enables live modifications!

- Mechanism similar to Lua (or possibly other script languages) embedded into a C or Ada application… but all works with a single language!

Advantages:

- **Compatibility**: Ada on both sides
- **Strong typing**: ranges, no type promotion, …
- **Composite types**: combinations of arrays and records
- Possibility of **sharing** some pieces of source code (typically type definitions) between the **HAC** program and the hosting program (of course, you have to freeze those sources).
- Programmers don't need to know more than one language