HAC Ada Compiler User Manual

https://github.com/zertovitch/hac
https://sourceforge.net/projects/hacadacompiler

Gautier de Montmollin (Krikos) gdemont@hotmail.com

Ed.	*Release*	*Comments*	
1	20201111	Initial release	sr
2	20201119	Add introduction and general organization	sr
6	20201121	Update document properties, HAC runtime & vector logo	sr
14	20201122	Add PDF documents properties and keywords, citations, update HAC runtime, progress review	sr
21	20201214	Add illustrations, new runtime functions, predefined types, HAC basic programming (to be translated)	sr
24	20201219	Add code examples in HAC runtime chapter, add exit code issue	sr
26	20201220	Add and populate chapter HAC firsts programs, translate first program	sr
27	20210303	Change hax to hac	sr
33	20210308	Runtime update, various improvements, reorganize chapters, fix many typos	sr
38	20210309	Final review for first public edition	sr
48	20210312	Change main title, Gautier email, replace many Hac typos, fix various typos, first release	sr
48			

Étapes de mise à jour du tableau d'historique. Avant toute modification du document :

- Positionner le curseur sur l'avant dernière ligne du tableau (celle au dessus de « Édition courante ») ;
- Créer une nouvelle ligne dans le tableau ;
- Sélectionner et copier la dernière ligne, de « Validation » à « Email » (tout sauf la première colonne) ;
- Positionner le curseur sur l'avant dernière ligne, dans la colonne « Validation » ;
- Coller ;

Reporter l'indice de la dernière ligne dans la nouvelle ligne.

Author

Gautier de Montmollin (Krikos) - gdemont@hotmail.com

Manual

Stéphane Rivière (Number Six) - stef@genesix.org
Gautier de Montmollin (Krikos) - gdemont@hotmail.com

The "Excuse me I'm French" speech - The main author of this manual is a Frenchman[1] with basic English skills. Frenchmen are essentially famous as frog eaters[2]. They have recently discovered that others ~~forms of communication~~ languages are widely used on earth. So, as a frog eater, I've tried to write some stuff in this foreign dialect loosely known here under the name of english. However, it's a well known fact, frogs don't really speak english. So your help is welcome to correct this bloody manual, for the sake of the wildebeests, and penguins too.

Edition

1 48 - 2021-03-12

[1] This do not apply to Gautier, author of HAC, who is a proud multiligual Swiss citizen ;)

[2] We could be famous as designers of the Concorde, Ariane rockets, Airbus planes or even Ada computer language but, definitely, Frenchmen have to wear beret with bread baguette under their arm to go eating frogs in a smokey tavern. That's *le cliché* :)

https://this-page-intentionally-left-blank.org

gdemont@hotmail.com

https://github.com/zertovitch/hac

Summary

Introduction

1 About HAC

HAC is a recursive acronym meaning HAC Ada Compiler. HAC isn't a native code compiler but a Virtual Machine compiler which comes with a very compact and monolithic run-time executor.

As the HAC author says: "HAC is perhaps the first open-source (albeit very partial) Ada compiler (and virtual machine interpreter) fully programmed in Ada itself. It wasn't written from scratch, but is based on a renovation of SmallAda, a system developed around 1990 and then abandoned."

2 HAC purposes

HAC can be used for small Ada sand-boxed prototypes, education and scripting.

As an education tool, HAC is an excellent Ada subset for programming introduction.

As a script language, thanks for its shebang handling and its useful environment functions, HAC is the most Ada compact and powerful script engine you ever dream for. The HAC compilation is straightforward. The executor is ridiculously small and fully standalone. Move the hac program to your path and voila!

Last but not the least, HAC sources are fully compatible with Ada compilers, through the compatibility package HAL !

2.1 Thanks

The authors of SmallAda, listed below, for making their work open-source.

Jean-Pierre Rosen for the free AdaControl tool which was very helpful detecting global variables stemming from SmallAda's code.
https://www.adalog.fr/en/adacontrol.html
https://sourceforge.net/projects/adacontrol

AdaCore for providing their excellent Ada compiler for free.
https://www.adacore.com/community

2.2 HAC history

2020: FOSDEM's Ada Developer Room: https://fosdem.org/2020/schedule/event/ada_hac
2013: January 24th: Day One of HAC: Hello World, Fibonacci and other tests work!
2009: A bit further trying to make the translation of SmallAda sources succeed (P2Ada was improved on the way, for WITH statements and modularity)...
1999: Automatic translation of Mac Pascal SmallAda sources o Ada, using P2Ada.
1989: SmallAda is derived from CoPascal; works only within two very system-dependent environments (a Mac GUI, a DOS GUI); two similar source sets in two Pascal dialects (Mac Pascal, Turbo Pascal).
1986: CoPascal (Schoening).
1970: Pascal-S (Wirth).

❑ Authors of SmallAda (in Pascal)

1990 Manuel A. Perez	Macintosh version
1990 Arthur V. Lopes	integrated environment for IBM-PC
1989 Arthur V. Lopes	window-oriented monitoring for IBM-PC
1988 Stuart Cramer and Jay Kurtz	refinement of tasking model
1987 Frederick C. Hathorn	conversion of CoPascal

❑ Author of CoPascal (derived from Niklaus Wirth's Pascal-S)

1986 Charles Schoening

❑ SmallAda sources

You can find the SmallAda sources and examples in the "archeology" folder. The Turbo Pascal sources files are smaller than the Mac Pascal ones, probably because of the memory constraints of the 16-bit DOS system. So the Turbo Pascal sources are especially cryptic and sparsely commented, full with magic numbers and 1-letter variables, many of them global.

3 Syntax notation

Inside a command line:
- A parameter between brackets [ ] is optional ;
- Two parameters separated by | are mutually exclusives.

<<<TODO>>>Add more structured syntax notation.

4 About Ada

Some general thoughts about Ada.

4.1 Introduction

This language is not known enough yet, at least not to the majority of us, much to the detriment of many potential users for that matter. Compared to the fashionable languages, Ada is more portable, more readable, allows for higher abstraction levels and has features and functionalities unseen in other languages. Ada also allows a more comfortable experience in system programming[3] and proves itself light enough to be usable on low class 8 bit processors[4].

Ada is the name of the first programmer to ever exist in humanity. And this first programmer was a woman: Augusta Ada Byron King, Countess of Lovelace, born in 1815, daughter of Byron, the great poet, Charles Babbage's assistant, she wrote programs destined to run on his famous machine.

Ada is an American military norm[5] as well as an international civil norm[6], it is the first object oriented language to be standardized at an international level. All Ada compilers must strictly adhere to the standard. There are hundreds of compilers destined to run on that many platforms but all of them will produce a code that runs identically.

Ada is used everywhere security is critical: Airbus (A3xx civil series and A400 military), Alsthom (High speed train), Boeing (777 and 787), EADS (Eurofighter, Ariane, ATV, many spaces probes), STS (line 14 Meteor), NASA (Electric power supply of the International Space Station). The list goes on and on. Everywhere reliability and security must come first, Ada is the language of choice.

4.2 Why use Ada

Ada was created because software engineering is a human activity. Humans make mistakes, the Ada compiler is friend to developers. Ada is also friend to project managers for large scale development. An Ada application is written, expanded and maintained very naturally. For these reasons, Ada is also friend to executives. Ada is the language of happy programmers, managers and users.

Because Ada is a comfortable language by it's expressiveness and a restful language by it's reliability, humans involved with Ada also reflect the image of their language. The Ada community is a very comfortable community to visit and most meetings are very enlighting. Free libraries are numerous and are usually of a very high quality. Finally, the Ada community is very highly active and by now growing again.

[3] Thanks to it's representation clauses that obliterates the need to use bit masking for XORed for bit manipulation. This functionality *essential to system programming* is simply not there in C or even in Assembly language.

[4] Components that have at their disposal a couple dozen bytes of RAM and a couple Kilobytes of programming memory.

[5] MIL-STD-1815

[6] ISO/IEC 8652

4.3 The ending word

When Boeing decided, two decades ago, that all software for the 777[7] would be exclusively written in Ada, the corporate associates of the constructor made the remark that they were using, for a long time, languages such as C, C++ and assembly language and that they were fully satisfied with them. Boeing simply answered that only firms that could provide Ada software would be considered in contracts offerings. Therefore, the firms converted themselves to Ada.

Today, the development of software for the Boeing 777 nicknamed « The Ada Plane », has been performed and it is essentially thanks to the very big commercial success of this plane that Boeing was able to maintain the revenues created by its civil activities.

And what do the Boeing partner firms do from now on ? They continue to develop their new software in none other than... Ada, and here's why:
- They noticed that the length of time to convert developers to Ada is usually rather short. In a week, the developer is comfortable enough to write software in Ada and in less than a month, he feels totally comfortable with the language ;
- These firms did their accounting: written in Ada, software costs less, present less anomalies, are ready sooner and are easier to maintain.

[7] The Boeing 777 is the world's biggest two engines plane and the first civil Boeing having electrical flight commands, ten years later the Airbus A320.

Getting started

One can write neatly in any language, including C. One can write badly in any language, including Ada. But Ada is the only language where it is more tedious to write badly than neatly.
Jean-Pierre Rosen

1 Distribution

1.1 Directories

src	sources of HAC, plus the compatibility package HAL
exm	examples
exm-manual	sources related to this manual
test	testing

1.2 Key files

Key files are located in the main directory.

hac.pdf	this file
hac.gpr	project file for building HAC with GNAT
hac_work.xls	workbook containing, notably, a bug list and a to-do list
save.adb	backup script, works both with HAC and GNAT!

2 Get a compiler

On Debian-based Linuxes like Ubuntu, GNAT is part of the standard packages.

For other systems, you can get GNAT from AdaCore's Web site: https://www.adacore.com/community or from other sites providing GCC (the GNU Compiler Collection) with Ada support.

Check http://www.getadanow.com for instance.

3 HAC build

3.1 Windows

❑ Windows compiler

<<<TODO>>>

❑ Compilation

Assuming you have the GNAT compiler installed, do the following from a command line interpreter. Open a terminal:

```
user@system: gnatmake -P hac
user@system: cd exm
user@system: ..\hac gallery.adb
```

Alternatively, with the help of a graphic file manager, you can go into the "exm" folder and double-click "e.cmd".

3.2 Linux

❑ Debian 10 compiler

Install an Ada compiler. Depending of your system, replace aptitude by apt-get or apt, preceded or not by sudo:

```
user@system: aptitude install gnat-8 gprbuild git
```

❑ Compilation

Assuming you wish to install HAC under /root, using it as a system tool:

Assuming you have the GNAT compiler installed, do the following from a command line interpreter. Open a terminal:

```
user@system: cd /root
user@system: git clone https://github.com/zertovitch/hac
user@system: cd hac
user@system: gprbuild -P hac
user@system: ln -s -f /root/hac/hac /usr/local/bin/hac

user@system: gnatmake -P hac
user@system: cd exm
user@system: ../hac gallery.adb
```

Alternatively, with the help of a graphic file manager, you can go into the "exm" folder and double-click "e.sh".

<<<TODO>>> make e.sh - or better e.adb :)

gdemont@hotmail.com
https://github.com/zertovitch/hac

HAC language

Investment in C programs reliability will increase up to exceed the probable cost of errors or until some-one insists on recoding everything in Ada.
Gilb's laws synthesis

1 Design points

HAC reads Ada sources from any stream. In addition to files, it is able to read from a Zip archive (plan is to have sets with many sources like libraries in Zip files, for instance), from an internet stream, from an editor buffer (see the LEA project), from a source stored in memory, etc.

One goal is to complement or challenge GNAT, for instance in terms of compilation speed, or object code compactness, or usability on certain targets or for certain purposes (like scripting jobs).

HAC could theoretically be also used for tuning run-time performance; this would require compiling on other targets than p-code, that is, real machine code for various platforms.

2 Language subset

HAC supports a very small subset of the Ada language. On the other hand, Ada is very large, so even a small subset could already fit your needs.

There is a short (and growing) list of small programs that are working (in the meaning: the compilation succeeds and the execution gives a correct output). They are listed in the project file hac_exm.gpr in the "exm" directory.

You will see that the "Pascal subset" plus tasking is more or less supported, so you have things like subprograms (including nested and recursive subprograms), expressions, that are working, for instance.

3 Predefined types

HAC handles Integer, Real, Character, String, VString & Boolean as predefined types.

3.1 Integer

An HAC integer is always 64 bits wide, whatever the host system.

3.2 Real

18 digits accuracy.

3.3 Character

<<<TODO>>>

3.4 String

<<<TODO>>>

3.5 VString

Unlimited length. This type is available in the HAL package.

3.6 Boolean

True or False.

4 General capabilities

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 Ada language has types depending on parameters whose value at compile-time or at run-time. A typical example is the predefined String type[8]. You can have s : String (1..10), a stack-allocated fixed string of length 10, but also s : String (1..n) where n is a function parameter or another dynamic value, or even s as a function parameter; the bounds are not explicit but can be queried.
But so far HAC doesn't support unconstrained types, one of Ada most powerful constructs.

However, the String type is implemented, in a very limited way. So far you can only define variables (or types, record fields) with it, like:

[8] The definition of the Ada language minimizes the "magic items", so String is also defined somewhere in Ada as:
`type String is array(Positive range <>) of Character`

```
Digitz : constant String (1..10) := "0123456789";
```

... and output them with Put, Put_Line.

For general string manipulation, the most practical way with the current versions of HAC is to use the VString variable-length string type.

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) are overloaded many times, like "Put", "Get", "+", "&", ...

Tasks are implemented, but not yet tested.

A more systematic testing is done in the "test" directory.

5 Keywords and special characters

5.1 Special characters

+ - * / ( ) [ ] , ; &

5.2 Reserved words

ABORT ABS ABSTRACT ACCEPT ACCESS ALIASED ALL AND ARRAY AT BEGIN BODY CASE CONSTANT DECLARE DELAY DELTA DIGITS DO ELSE ELSIF END ENTRY EXCEPTION EXIT FOR FUNCTION GENERIC GOTO IF IN INTERFACE IS LIMITED LOOP MOD NEW NOT NULL OF OR OTHERS OUT OVERRIDING PACKAGE PRAGMA PRIVATE PROCEDURE PROTECTED RAISE RANGE RECORD REM RENAMES REQUEUE RETURN REVERSE SELECT SEPARATE SOME SUBTYPE SYNCHRONIZED TAGGED TASK TERMINATE THEN TYPE UNTIL USE WHEN WHILE WITH XOR

6 Metrics

Object code size:
Stack size: 1 000 000 elements
Simultaneous identifiers: 10000

HAC runtime

There are 10 types of people in the world: those who understand binary and those who don't.
Anonymous

1 Introduction

The HAC runtime is located in the ./src/hal.ads (specification file), ./src/hal.adb and ./src/hal-non_standard.adb files.

The specification file lists all the functions and procedures available.

HAL being the only available package so far, HAC programs need to have, at their very top:

```
with HAL;
```

2 Real functions

Floating-point numeric type functions.

2.1 Sin

- Description

Performs sine operation with arguments in radians.

- Usage

function Sin (F : Real) return Real

- Example

<<<TODO>>>

2.2 Cos

- Description

Performs cosine operation with arguments in radians.

- Usage

function Cos (F : Real) return Real

- Example

<<<TODO>>>

2.3 Arctan

- Description

Performs arc-tangent operation with arguments in radians.

- Usage

function Arctan (F : Real) return Real

- Example

<<<TODO>>>

2.4 Log

- Description

Performs natural logarithm operation.

- Usage

function Log (F : Real) return Real

- Example

<<<TODO>>>

2.5 Exp

- Description

Performs exponential operation.

- Usage

function Exp (F : Real) return Real

- Example

<<<TODO>>>

2.6 Sqrt

- Description

performs square root operation.

- Usage

function Sqrt (I : Integer) return Real
function Sqrt (F : Real) return Real

2.7 Round

- Description

Performs rounding operation from Real to an Integer.

- Usage

function Round (F : Real) return Integer

- Example

<<<TODO>>>

2.8 Truncate

- Description

Performs truncating operation from Reafl to an Integer.

- Usage

function Trunc (F : Real) return Integer

- Example

<<<TODO>>>

3 Real operators

3.1 **

- Description

Performs F1 ^ F2 (power of) operation.

- Usage

function "**" (F1, F2 : Real) return Real

- Example

<<<TODO>>>

4 Randomize functions

4.1 Rand

- Description

Returns random Integer number in the real range [0, 1+1[, truncated to lowest integer.

- Usage

function Rand (I : Integer) return Integer

- Example

Rand (10) returns equiprobable integer values between 0 and 10 (so, there are 11 possible values).

4.2 Rnd

- Description

Returns random Real number from 0 to 1, uniform.

- Usage

function Rnd return Real

- Example

<<<TODO>>>

5 Characters functions

5.1 Succ

- Description

Returns next character in ASCII table order.

- Usage

function Succ (C : Character) return Character

- Example

<<<TODO>>>

5.2 Pred

- Description

Returns previous character in ASCII table order.

- Usage

function Pred (C : Character) return Character

- Example

<<<TODO>>>

6 VString functions

Variable-size string type

Null_VString : VString

6.1 To_Lower

- Description

Convert a Character or a VString to lower case.

- Usage

function To_Lower (Item : Character) return Character
function To_Lower (Item : VString) return VString

- Example

<<<TODO>>>

6.2 To_Upper

- Description

Convert a Character or a VString to upper case.

- Usage

function To_Upper (Item : Character) return Character
function To_Upper (Item : VString) return VString

- Example

<<<TODO>>>

6.3 Element

- Description

Return the Character in Index position of the Vstring argument.
Index starts at one.

- Usage

function Element (Source : VString; Index : Positive) return Character

- Example

<<<TODO>>>

6.4 Ends_With

- Description

Check if Vstring Item ends with another VString or String Pattern.

- Usage

function Ends_With (Item : VString; Pattern : String) return Boolean
function Ends_With (Item : VString; Pattern : VString) return Boolean

- Example

```
- Check VString with String pattern
if Ends_With (+"package",  "age") then
  Put_Line ("Match !");
end if;

- Check VString with VString pattern
if Ends_With (+"package",  +"age") then
  Put_Line ("Match !");
end if;
```

6.5 Head

- Description

Extract a VString between the beginning to Count Value to a VString.
Count starts at one.

- Usage

function Head (Source : VString; Count : Natural) return VString

- Example

```
Put_Line (Head (+"ABCDEFGH", 4));
```

```
"ABCD"
```

6.6 Index

- Description

Returns Natural start position of String or VString Pattern in the target Vstring Source, From a starting index.
Natural is zero if not found.
Natural starts at one.

- Usage

function Index (Source : VString; Pattern : String) return Natural
function Index (Source : VString; Pattern : VString) return Natural
function Index (Source : VString; Pattern : String; From : Natural) return Natural
function Index (Source : VString; Pattern : VString; From : Natural) return Natural

- Example

```
if Index (+"ABCDABCD", +"BC") = 2 then
    Put_Line ("Match !");
end if;

if Index (+"ABCDEFGH", +"BC", 4) = 6 then
    Put_Line ("Match !");
end if;
```

6.7 Index_backward

- Description

From the end of the target Vstring Source, returns Natural start position of String or VString Pattern in the target Vstring Source, From a backward starting index.
Natural is zero if not found.
Natural starts at one.

- Usage

function Index_Backward (Source : VString; Pattern : String) return Natural
function Index_Backward (Source : VString; Pattern : VString) return Natural
function Index_Backward (Source : VString; Pattern : String; From : Natural) return Natural
function Index_Backward (Source : VString; Pattern : VString; From : Natural) return Natural

- Example

```
if Index_Backward (+"abcdefabcdef", +"cd") = 9 then
    Put_Line ("Match !");
end if;

if Index_Backward (+"abcdefabcdef", +"cd", 8) = 3 then
    Put_Line ("Match !");
end if;
```

6.8 Length

- Description

The function Length returns the length of the VString represented by Source.

- Usage

function Length (Source : VString) return Natural

- Example

```
Put (Length (+"ABCDEFGH"));

8
```

6.9 Slice

- Description

The function Slice returns a Vstring portion of the Vstring represented by Source delimited by From and To.
From and To start at one.

- Usage

function Slice (Source : VString; From : Positive; To : Natural) return VString

- Example

```
Put_Line (Slice (+"ABCDEFGH", 2,4));

"BCDE"
```

6.10 Tail

- Description

Extract a VString from Source between its end to backward Count Value.
Count starts at one (backward).

- Usage

function Tail (Source : VString; Count : Natural) return VString

- Example

```
Put_Line (Tail (+"ABCDEFGH", 4));

"EFGH"
```

6.11 Tail_After_Match

- Description

Extract a VString from Source starting from Pattern+1 position to the end.

- Usage

function Tail_After_Match (Source : VString ; Pattern : VString) return VString

- Examples

```
Put_Line (Tail_After_Match (Path, '/'));
"gnx-startup"

Put_Line (Tail_After_Match (Path, "ix"));
"/gnx-startup"

Put_Line (Tail_After_Match (Path, "gene"));
"six/gnx-startup"

Put_Line (Tail_After_Match (Path, "etc/genesix/gnx-startu"));
"p"

Put_Line (Tail_After_Match (Path, "/etc/genesix/gnx-startu"));
"p"

Put_Line (Tail_After_Match (Path, "/etc/genesix/gnx-startup"));
empty string

Put_Line (Tail_After_Match (Path, +"/etc/genesix/gnx-startupp"));
empty string

Put_Line (Tail_After_Match (Path, +"/etc/geneseven"));
empty string
```

6.12 Starts_With

- Description

Check if Vstring Item starts with another VString or String Pattern.

- Usage

function Starts_With (Item : VString; Pattern : String) return Boolean
function Starts_With (Item : VString; Pattern : VString) return Boolean

- Example

```
- Check VString with String pattern
if Ends_With (+"package",  "pac") then
  Put_Line ("Match !");
end if;

- Check VString with VString pattern
if Ends_With (+"package",  +"pac") then
  Put_Line ("Match !");
end if;
```

6.13 Trim_Left

- Description

Returns a trimmed leading spaces VString of VString Source.

- Usage

function Trim_Left (Source : VString) return VString

- Example

```
Put_Line (Trim_Left (+"   ABCD   "));

"ABCD   "
```

6.14 Trim_Right

- Description

Returns a trimmed trailing spaces VString of VString Source.

- Usage

function Trim_Right (Source : VString) return VString

- Example

```
Put_Line (Trim_Right (+"   ABCD   "));

"   ABCD"
```

6.15 Trim_Both

- Description

Returns an all trimmed spaces VString of VString Source.

- Usage

function Trim_Both (Source : VString) return VString

- Example

```
Put_Line (Trim_Right (+"   AB CD   "));
"AB CD"
```

7 VString operators

7.1 +

- Description

Cast a String to a VString.

- Usage

function "+" (S : String) return VString

7.2 *

- Description

Duplicate a Character, String or VString Num times to a VString.

- Usage

function "*" (Num : Natural; Pattern : Character) return VString
function "*" (Num : Natural; Pattern : String) return VString
function "*" (Num : Natural; Pattern : VString) return VString

- Example

```
Put_Line (3 * "0");
"000"
Put_Line (3 * +"12");
"121212"
```

7.3 &

- Description

Concatenate a VString with a VString, String, Character, Integer and Real to a VString

- Usage

function "&" (V1, V2 : VString) return VString

function "&" (V : VString; S : String) return VString
function "&" (S : String; V : VString) return VString

function "&" (V : VString; C : Character) return VString
function "&" (C : Character; V : VString) return VString

function "&" (I : Integer; V : VString) return VString
function "&" (V : VString; I : Integer) return VString

function "&" (R : Real; V : VString) return VString
function "&" (V : VString; R : Real) return VString

7.4 =

- Description

Test equality between a VString and another VString or String.

- Usage

function "=" (Left, Right : VString) return Boolean
function "=" (Left : VString; Right : String) return Boolean

- Example

<<<TODO>>>

7.5 <

- Description

<<<TODO>>>

- Usage

function "<" (Left, Right : VString) return Boolean
function "<" (Left : VString; Right : String) return Boolean

- Example

<<<TODO>>>

7.6 <=

- Description

<<<TODO>>>

- Usage

function "<=" (Left, Right : VString) return Boolean
function "<=" (Left : VString; Right : String) return Boolean

- Example

<<<TODO>>>

7.7 >

- Description

<<<TODO>>>

- Usage

function ">" (Left, Right : VString) return Boolean
function ">" (Left : VString; Right : String) return Boolean

- Example

<<<TODO>>>

7.8 >=

- Description

- Usage

function ">=" (Left, Right : VString) return Boolean
function ">=" (Left : VString; Right : String) return Boolean

- Example

<<<TODO>>>

8 Type conversion functions

8.1 Chr

- Description

Convert an Integer to a Character.

- Usage

function Chr (I : Integer) return Character

- Example

<<<TODO>>>

8.2 Ord

- Description

Convert a Character to an Integer

- Usage

function Ord (C : Character) return Integer

- Example

<<<TODO>>>

8.3 To_VString

- Description

Convert a Char or a String type into VString type.

- Usage

function To_VString (C : Char) return VString
function To_VString (S : String) return VString

- Example

```
Input : String := "ABC";
Result : VString;
Result := To_VString (Input);
```

8.4 Image

- Description

Image returns a VString representation of Integer, Real, Ada.Calendar.Time & Duration.

- Usage

function Image (I : Integer) return VString
function Image (F : Real) return VString
function Image_Attribute (F : Real) return VString
function Image (T : Ada.Calendar.Time) return VString
function Image (D : Duration) return VString

- Example

<<<TODO>>>

8.5 Image_Attribute

- Description

Image_Attribute returns an Real VString image "as is" of F, instead of Image (F : Real) which returns a "nice" VString image of F

- Usage

Image_Attribute (F : Real) returns VString

- Example

```
Put_Line (Image (Real (4.56789e10)));
45678900000.0
Put_Line (Image_Attribute (4.56789e10));
4.56789000000000E+10
```

8.6 Integer_Value

- Description

Convert a VString to an integer.

- Usage

function Integer_Value (V : VString) return Integer

- Example

<<<TODO>>>

8.7 Float_Value

- Description

Convert a VString to a Real.

- Usage

function Float_Value (V : VString) return Real

- Example

<<<TODO>>>

9 Console i/o functions

HAC comes with a real console/terminal input where several inputs can be made on the same line, followed by a "Return". It behaves like for a file. Actually it *could* be a file, if run like this:

```
user@system: prog <input.txt
```

9.1 Get_Needs_Skip_Line

- Description

- Usage

function Get_Needs_Skip_Line return Boolean is (True)

- Example

<<<TODO>>>

9.2 Get

- Description

- Usage

procedure Get (C : out Character) renames Ada.Text_IO.Get
procedure Get (I : out Integer)
procedure Get (F : out Real)
procedure Get (S : out String)

- Example

<<<TODO>>>

9.3 Get_Immediate

- Description
- Usage

procedure Get_Immediate (C : out Character)

- Example

```
procedure Pause is

Dummy : Character;

begin

  Put_Line (+"Press any key to continue...");
  Get_Immediate(Dummy);

end Pause;
```

9.4 Get_Line

- Description

Get and then move file pointer to next line (Skip_Line)

- Usage

procedure Get_Line (C : out Character)
procedure Get_Line (I : out Integer)
procedure Get_Line (F : out Real)
procedure Get_Line (V : out VString)

- Example

<<<TODO>>>

9.5 Skip_Line

- Description

Clear the current line and gets ready to read the line after it, or skip more when Spacing is passed.

- Usage

procedure Skip_Line (File : File_Type; Spacing : Positive)

- Example

<<<TODO>>>

9.6 Put

- Description

<<<TODO>>>

- Usage

```
procedure  Put (C : Character)
procedure  Put (I : Integer;
                Width : Ada.Text_IO.Field := IIO.Default_Width;
                Base : Ada.Text_IO.Number_Base := IIO.Default_Base);
procedure  Put (F : Real;
                Fore  : Integer := RIO.Default_Fore;
                Aft : Integer := RIO.Default_Aft;
                Expo : Integer := RIO.Default_Exp);
procedure  Put (B : Boolean;
                Width : Ada.Text_IO.Field := BIO.Default_Width);
procedure  Put (S :  String);
procedure  Put (V :  VString);
```

- Example

<<<TODO>>>

9.7 Put_Line

- Description

Put and then New_Line.

- Usage

```
procedure Put_Line (C : Character);
procedure Put_Line (I : Integer; Width : Ada.Text_IO.Field := IIO.Default_Width;
                    Base : Ada.Text_IO.Number_Base := IIO.Default_Base);
procedure Put_Line (F : Real; Fore : Integer := RIO.Default_Fore;
                    Aft : Integer := RIO.Default_Aft; Expo : Integer := RIO.Default_Exp);
procedure Put_Line (B : Boolean; Width : Ada.Text_IO.Field := BIO.Default_Width);
procedure Put_Line (S : String);
procedure Put_Line (V : VString);
```

- Example

<<<TODO>>>

9.8 New_Line

- Description

Create a new blank line, or more than one when Spacing is passed.

- Usage

procedure New_Line (Spacing : Positive)

- Example

<<<TODO>>>

9.9 End_Of_File

- Description

Return True if the end of file is reached.

- Usage

function End_Of_File return Boolean

- Example

```
Open (File_Tmp_Handle, +"./toto");
while not End_Of_File (File_Tmp_Handle) loop
  Get_Line (File_Tmp_Handle, Line_Buffer);
end loop;
Close (File_Tmp_Handle);
```

9.10 End_Of_Line

- Description

Return True if the end of line is reached.

- Usage

function End_Of_Line return Boolean

- Example

<<<TODO>>>

10 File i/o functions

subtype File_Type is Ada.Text_IO.File_Type;

10.1 Open

- Description

Open a file.
File mode is “In” (read mode).

<<<TODO>>> Comment File_Type

- Usage

procedure Open (File : in out File_Type; Name : String)
procedure Open (File : in out File_Type; Name : VString)

- Example

```
Open (File_Tmp_Handle, +"./toto");
while not End_Of_File (File_Tmp_Handle) loop
  Get_Line (File_Tmp_Handle, Line_Buffer);
end loop;
Close (File_Tmp_Handle);
```

10.2 Create

- Description

Create a file.
File mode is “Out” (write mode).

- Usage

procedure Create (File : in out File_Type; Name : String)
procedure Create (File : in out File_Type; Name : VString)

- Example

```
Create (File_Tmp_Handle, +"./toto");
while not End_Of_File (File_Tmp_Handle) loop
  Get_Line (File_Tmp_Handle, Line_Buffer);
end loop;
Close (File_Tmp_Handle);
```

10.3 Append

- Description

Append a file.

File mode is “Out” (write mode).

- Usage

procedure Append (File : in out File_Type; Name : String)
procedure Append (File : in out File_Type; Name : VString)

- Example

```
Append (File_Tmp_Handle, +"./toto");
while not End_Of_File (File_Tmp_Handle) loop
  Get_Line (File_Tmp_Handle, Line_Buffer);
end loop;
Close (File_Tmp_Handle);
```

10.4 Close

- Description

Close a file.

- Usage

procedure Close (File : in out File_Type)

- Example

```
Open (File_Tmp_Handle, +"./toto");
while not End_Of_File (File_Tmp_Handle) loop
  Get_Line (File_Tmp_Handle, Line_Buffer);
end loop;
Close (File_Tmp_Handle);
```

10.5 Get

- Description

Get the current line.

- Usage

procedure Get (File : File_Type; C : out Character)
procedure Get (File : File_Type; S : out String)
procedure Get (File : File_Type; I : out Integer);
procedure Get (File : File_Type; F : out Real);

- Example

```
Create (File_Tmp_Handle, +"./toto");

while not End_Of_File (File_Tmp_Handle) loop

  Get (File_Tmp_Handle, Line_Buffer);
  Skip_Line;

end loop;

Close (File_Tmp_Handle);
```

10.6 Get_Line

- Description

Get the current line and then move file pointer to the next line.

- Usage

procedure Get_Line (File : File_Type; C : out Character)
procedure Get_Line (File : File_Type; I : out Integer)
procedure Get_Line (File : File_Type; F : out Real)
procedure Get_Line (File : File_Type; V : out VString)

- Example

```
Create (File_Tmp_Handle, +"./toto");

while not End_Of_File (File_Tmp_Handle) loop

  Get_Line (File_Tmp_Handle, Line_Buffer);

end loop;

Close (File_Tmp_Handle);
```

10.7 Skip_Line

- Description

Clear the current line and gets ready to read the line after it, or skip more when Spacing is passed.

- Usage

procedure Skip_Line (File : File_Type; Spacing : Positive)

- Example

```
Create (File_Tmp_Handle, +"./toto");

while not End_Of_File (File_Tmp_Handle) loop

  Get (File_Tmp_Handle, Line_Buffer);
```

```
   Skip_Line;
end loop;
Close (File_Tmp_Handle);
```

10.8 Put

- Description

<<<TODO>>>

- Usage

```
procedure Put (File : File_Type; C : Character)
procedure Put (File : File_Type
               I : Integer;
               Width : Ada.Text_IO.Field := IIO.Default_Width;
               Base : Ada.Text_IO.Number_Base := IIO.Default_Base)
procedure Put (File : File_Type;
               F : Real;
               Fore : Integer := RIO.Default_Fore;
               Aft : Integer := RIO.Default_Aft;
               Expo : Integer := RIO.Default_Exp)
procedure Put (File  : File_Type; B : Boolean; Width : Ada.Text_IO.Field := BIO.Default_Width)
procedure Put (File : File_Type; S : String);
procedure Put (File : File_Type; V : VString);
```

- Example

<<<TODO>>>

10.9 Put_Line

- Description

Put and then New_Line (for S: it is the same as Ada.Text_IO.Put_Line)

- Usage

```
procedure  Put_Line (File : File_Type; C : Character);
procedure  Put_Line (File  : File_Type;
                     I     : Integer;
                     Width : Ada.Text_IO.Field       := IIO.Default_Width;
                     Base  : Ada.Text_IO.Number_Base := IIO.Default_Base);
procedure  Put_Line (File : File_Type;
                     F    : Real;
                     Fore : Integer := RIO.Default_Fore;
                     Aft  : Integer := RIO.Default_Aft;
                     Expo : Integer := RIO.Default_Exp);
procedure  Put_Line (File  : File_Type;
```

```
                B     : Boolean;
                Width : Ada.Text_IO.Field := BIO.Default_Width);
procedure  Put_Line (File : File_Type; S : String);
procedure  Put_Line (File : File_Type; V : VString);
```

- Example

<<<TODO>>>

10.10 New_Line

- Description

Create a new blank line, or more than one when Spacing is passed.

- Usage

procedure New_Line (File : File_Type; Spacing : Positive)

- Example

<<<TODO>>>

10.11 End_Of_Line

- Description

Return true if end of line is reached.

- Usage

function End_Of_Line (File : File_Type) return Boolean
function End_Of_Line (File : File_Type) return Boolean

- Example

<<<TODO>>>

10.12 End_Of_File

- Description

Return true if end of file is reached.

- Usage

function End_Of_File (File : File_Type) return Boolean
function End_Of_File (File : File_Type) return Boolean

- Example

<<<TODO>>>

10.13 Error handling

Input/output file errors are managed by HAC by means of an orderly exit :

```
HAC VM: raised Status_Error
File already open
Trace-back locations:
file_append.adb: File_Append.Nested_1.Nested_2 at line 12
file_append.adb: File_Append.Nested_1 at line 23
file_append.adb: File_Append at line 26
```

11 Time functions

subtype Time is Ada.Calendar.Time;

11.1 Clock

- Description

Returns current Time at the time it is called.

- Usage

function Clock return Time

- Example

```
function Time_Stamp return VString is

Current_Time : constant Time := Clock;
Day_Secs, Day_Mins : Integer;

begin

  Day_Secs := Integer (Seconds (Current_Time));
  Day_Mins := Day_Secs / 60;

  return Image (Year (Current_Time)) &
         Time_Format (Month (Current_Time)) &
         Time_Format (Day (Current_Time)) &
         & "-" &
         Time_Format (Day_Mins / 60) &
         Time_Format (Day_Mins mod 60) &
         Time_Format (Day_Secs mod 60);

end Time_Stamp;
```

11.2 Year

- Description

Returns year from Date.

- Usage

function Year (Date : Time) return Integer

- Example

See Clock function above.

11.3 Month

- Description

Return Month from Date.

- Usage

function Month (Date : Time) return Integer

- Example

See Clock function above.

11.4 Day

- Description

Return Day from Date.

- Usage

function Day (Date : Time) return Integer

- Example

See Clock function above.

11.5 Seconds

- Description

Return Seconds from Date.

- Usage

function Seconds (Date : Time) return Duration

- Example

See Clock function above.

12 Time operators

12.1 -

- Description

Return a Duration substracting Right from Left times.

- Usage

function "-" (Left : Time; Right : Time) return Duration

- Example

<<<TODO>>>

13 Semaphore functions

Specific stuff from SmallAda.

13.1 Wait

- Description

<<<TODO>>>

- Usage

procedure Wait (S : Semaphore);

- Example

<<<TODO>>>

13.2 Signal

- Description

<<<TODO>>>

- Usage

procedure Signal (S : Semaphore);

- Example

<<<TODO>>>

14 System

14.1 Argument_Count

- Description

Count arguments passed to the current program.

- Usage

function Argument_Count return Natural

- Example

```
procedure Arguments is
begin
  Put_Line (Command_Name);
  Put_Line (Argument_Count);

  for A in 1 .. Argument_Count loop
    Put_Line ("  --> [" & Argument (A) & ']');
  end loop;
end Arguments;
```

14.2 Argument

- Description

Returns Argument of index Number.

- Usage

function Argument (Number : Positive) return VString

- Example

See Argument_Count above.

14.3 Command_Name

- Description

Returns full qualified current program name.

- Usage

function Command_Name return VString

- Example

```
user@system: hac gnx-instance

Put_Line (Command_Name);
```

```
/home/sr/Seafile/Sowebio/informatique/dev/ada/app/gnx/src/gnx-instance
```

14.4 Set_Exit_Status

- Description

Returns exit code to system.

- Usage

procedure Set_Exist_Status (Code : in Integer)

- Example

<<<TODO>>>

14.5 Get_Env

- Description

Returns environment variable Name.

- Usage

function Get_Env (Name : String) return VString
function Get_Env (Name : VString) return VString

- Example

<<<TODO>>>

14.6 Set_Env

- Description

Set environment variable Name.

- Usage

procedure Set_Env (Name : String; Value : String)
procedure Set_Env (Name : VString; Value : String)
procedure Set_Env (Name : String; Value : VString)
procedure Set_Env (Name : VString; Value : VString)

- Example

<<<TODO>>>

14.7 Current_Directory

- Description

Returns current directory.

- Usage

function Current_Directory return VString

- Example

<<<TODO>>>

14.8 Set_Directory

- Description

Change to an existing directory Directory.

- Usage

procedure Set_Directory (Directory : String)
procedure Set_Directory (Directory : VString)

- Example

<<<TODO>>>

14.9 Copy_File

- Description

Copy a file Source_Name to a file Target_Name

- Usage

procedure Copy_File (Source_Name : String; Target_Name : String)
procedure Copy_File (Source_Name : VString; Target_Name : String)
procedure Copy_File (Source_Name : String; Target_Name : VString)
procedure Copy_File (Source_Name : VString;Target_Name : VString)

- Example

<<<TODO>>>

14.10 Delete_File

- Description

Delete a file Name.

- Usage

procedure Delete_File (Name : String)
procedure Delete_File (Name : VString)

- Example

<<<TODO>>>

14.11 Exists

- Description

Returns True if file or directory Name exists.

- Usage

function Exists (Name : String) return Boolean
function Exists (Name : VString) return Boolean;

- Example

```
if Exists (HAC_Dir & "/hac") then
   Put_Line ("HAC installation is done :)");
end if;
```

14.12 Directory_Exists

- Description

Returns True if directory Name exists.

- Usage

function Directory_Exists (Name : String) return Boolean
function Directory_Exists (Name : VString) return Boolean;

- Example

```
if Directory_Exists (HAC_Dir) then
   Put_Line ("HAC directory exists");
end if;
```

14.13 File_Exists

- Description

Returns True if file Name exists.

- Usage

function File_Exists (Name : String) return Boolean
function File_Exists (Name : VString) return Boolean;

- Example

```
if File_Exists (HAC_Dir & "/hac") then
   Put_Line ("HAC interpreter is build :)");
end if;
```

14.14 Rename

- Description

Renames a file or a directory from Old_Name to New_Name.

- Usage

procedure Rename (Old_Name : String; New_Name : String)
procedure Rename (Old_Name : VString; New_Name : String)
procedure Rename (Old_Name : String; New_Name : VString)
procedure Rename (Old_Name : VString; New_Name : VString)

- Example

<<<TODO>>>

14.15 Shell_Execute

- Description

Executes Command returning the errorlevel from executed Command if needed.

- Usage

procedure Shell_Execute (Command : String)
procedure Shell_Execute (Command : VString)
procedure Shell_Execute (Command : String; Result out Integer)
procedure Shell_Execute (Command : VString; Result out Integer)

- Example

Without exit code handling :

```
Shell_Execute ("upx " & HAC_Dir & "/hac");
```

With exit code handling :

```
Shell_Execute ("ln -s -f " & HAC_Dir & "/hac /usr/local/bin/hac", Err);

if Err /= 0 then
  Put_Line (+"Error, symbolink link not created");
end if;
```

14.16 Directory_Separator

- Description

Returns system directory separator.

- Usage

function Directory_Separator return Character

- Example

With an Unix or Unix like system :

```
Put_Line (Directory_Separator);
/
```

HAC architecture

Doubling the number of programmers on a late project does not make anything else than double the delay.
Second Brook's Law

1 Introduction

<<<TODO>>> Short intro about compilers, interpreters, runtimes and executors, mainly some definitions to ease the reader.

2 Architecture

<<<TODO>>> Architecture overview.

HAC FAQ

The last bug isn't fixed until the last user is dead.
Sidney Markowitz

1 Return codes

HAC returns:
- 0 if the execution was completed without exception;
- 1 if an exception occurs during execution.

2 How to compile HAC programs with a “full Ada” compiler

2.1 HAC example

For many reasons, you may wish to compile a HAC program with a native-code Ada compiler like GNAT. For example to deeply speed execution time, even though HAC is a quite fast interpreter.

Create hac_to_gnat.adb:

```
with HAL; use HAL;

procedure HAC_To_GNAT is

Start : Integer := Integer (Seconds (Clock));

begin

  New_Line;
  Put_Line ("Process 500M iterations - each * is 1M iterations");
  New_Line;

  for I in 1 .. 500_000_000 loop
    if (I mod 1_100_000 = 0) then
      Put ("*");
    end if;
  end loop;
```

```
   New_Line;
   New_Line;
   Put_Line ("Process time: " & Trim_Left (Image (Integer (Seconds (Clock)) - Start)) & "s");
   New_Line;

end HAC_To_GNAT;
```

Then execute it the HAC way:

```
user@system: hac hac_to_gnat.adb

Process 500M iterations - each * is 1M iterations

**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**********

Process time: 672s
```

2.2 GNAT compilation

Then compile hac_to_gnat .adb:

```
user@system: gnatmake hac_to_gnat -I/etc/genesix/hac/src

gcc -c -I/etc/genesix/hac/src hac_to_gnat.adb
gcc -c -I./ -I/etc/genesix/hac/src -I- /etc/genesix/hac/src/hal.adb
gnatbind -I/etc/genesix/hac/src -x hac_to_gnat.ali
gnatlink hac_to_gnat.ali
```

And execute it:

```
user@system: ./hac_to_gnat

Process 500M iterations - each * is 1M iterations

**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**********

Process time: 2s
```

These timings were obtained on a Intel(R) Xeon(R) CPU D-1521 @ 2.40GHz running HAC 0.091 and GNAT Community 2020 (20200429-93) under GNU/Linux Debian 10.

Seems HAC is slow, but HAC compiles to a Virtual Machine which is interpreted. HAC is not a native-code compiler. How HAC compares with common shell scripting languages like Bash ? The answer’s below !

3 How fast is HAC ?

3.1 Timings compared to Bash

The same program written in Bash can't be executed *because it failed to allocate 500M elements*:

```
user@system: ./hac_to_bash

Process 500M iterations - each * is 1M iterations

./hac_to_bash: ligne 6: expansion des accolades : échec lors de l'allocation mémoire pour
500000000 éléments
./hac_to_bash: ligne 8: {1..500000000} %1000000 : erreur de syntaxe : opérande attendu (le symbole
erroné est « {1..500000000} %1000000 »)
```

So at 50M iterations, Bash was *still unable to execute the script*, ending with a cryptic "Process stopped" message. Finally, at 5M elements, after 19 seconds (no kidding) used to allocate elements, execution begins:

```
user@system: time ./hac_to_bash

Process 5M iterations - each * is 10K iterations

**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**********

real            0m42,542s
user            0m31,988s
sys             0m7,000s
```

Using this source:

```
echo ""
echo "Process 5M iterations - each * is 10K iterations"
echo ""

for i in {1..5000000}; do

  if [ $(($i %10000)) == 0 ]; then
     echo -n "*"
  fi

done

echo ""
```

Reducing hac_to_gnat.adb to 5M elements to easily compare execution times:

```
user@system: hac hac_to_gnat_5M.adb

Process 5M iterations - each * is 10K iterations

**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
**************************************************************************************************
```

```
**********************************************************************************************
**********

Process time: 6s
```

3.2 Conclusion

➪ This demonstrates HAC is not only an interpreter coming with *extensive capabilities and native strong typing*, but that *HAC it's seven times faster than Bash* too.

4 How to add a runtime procedure

<<<TODO>>> Explain and justify each steps.

https://github.com/zertovitch/hac/commit/56880520761 04a6ec639b680fedbced2dfa4ce20

5 How to add a runtime function

<<<TODO>>> Explain and justify each steps.

https://github.com/zertovitch/hac/commit/707ebcd1a2010d83c32175127 05a4bd71bbf31cd

6 How to add bound checking to a runtime function

<<<TODO>>> Explain and justify each steps.

https://github.com/zertovitch/hac/commit/4b5154f117e146e41353 1aafd1b0b604f2ca878c

Programs examples

Weinberg's Second Law : If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
Gerald Weinberg

1 HAC firsts programs

Here are some very simple programs in HAC, in order to familiarize yourself with the syntax of Ada. They come mainly from the "Algorithmic-Programming level A" UV of the CNAM[9] and are copyrighted by the CNAM and the author François Barththélemy. Their reproduction without modification is authorized for non-commercial use.

According to our own experience, the indentation and presentation of programs have been significantly improved and some programs have been added, such hellothree.adb

Theses programs are located in ./exm-firsts-programs.

1.1 A salute to the world

Create the file hello.adb:

```
with HAL; use HAL;  -- Always start with theses mandatory definitions
procedure Hello is - Program's name
begin
```

[9] French Conservatoire National des Arts & Métiers - https://www.cnam.fr

```
  Put_Line ("Salute to the World :)"); -- Display the string and go to the next line
  New_Line; -- Line feed

end Hello;
```

Run it:

```
user@system: hac hello

Salute to the World:)
```

1.2 A simpler way to run HAC programs

To simplify things, at least in Unix based system, rename hello.adb without extension and make it executable:

```
user@system: mv hello.adb hello

user@system: chmod +x hello
```

Add a shebang at the very first line of the program:

```
#!/bin/env hac

with HAL; use HAL;  -- Always start with theses mandatory definitions

procedure Hello is - Program's name

begin

  Put_Line ("Salute to the World :)"); -- Display the string and go to the next line
  New_Line; -- Line feed

end Hello;
```

Verify hello is now a standalone program:

```
user@system: hello

Salute to the World :)
```

1.3 A salute without use clause

You can omit the use clause but need then to prefix the procedures with the package name HAL. This is just for information and probably not the way to go for simple scripts :

```
#!/bin/env hac

with HAL;  --

procedure Hello is - Program's name

begin

  HAL.Put_Line ("Salute to the World :)"); -- Display the string and go to the next line
  HAL.New_Line; -- Line feed
```

```
end Hello;
```

<<<TODO>>> Chapter remaining to be translated and/or finished.

2 HAC examples

The program "gallery.adb" will run a bunch of demos that are located in the "exm" directory.
You can test HAC on any other example of course (the "*.adb" files in the "exm" and "test" directories).

As a bonus, you can build some examples with GNAT to compare the output. You can do it easily with the hac_exm.gpr project file. Since hac_exm.gpr is a text file, you can see there the progress (or the lack thereof) in the pieces of Ada code that are really working with HAC. See the "Limitations" section below as well.

<<<TODO>>> Lists ./exm examples programs with comments.

3 HAC Advents of Code

Advents of code (https://adventofcode.com/2020)

https://adventofcode.com/2020/about

https://github.com/zertovitch/hac/tree/master/exm/aoc/2020

<<<TODO>>> Lists programs with comments.

Basic programming

My Operating System is Emacs and Windows is its driver.
Anonymous

HAC is a scripting, prototyping and educational procedural Ada subset language. It is perfect to write small, medium and - why not ? - huge non object programs in a modular and structured way.

If your are not an experienced programmer mastering a method programming as a tool, you may find this chapter useful. Your creative spirit's the limit.

<<<TODO>>> Chapter remaining to be translated.

Notes

With the Wildebeest and the Penguin, there's no Bull.
Number Six

1 To-do list

1.1 HAC

The to-do list is located in the spreadsheet "To do", within the workbook "hac_work.xls".

1.2 Doc

Alphabetically reorder procedures and functions in HAC runtime instead of “hal.ads source ordered”.
Populate examples in HAC runtime chapter.
Translate Basic programming chapter.
Hunt <<<TODO>>> tags :)

2 Issues

2.1 Github

Issues are listed on Github: https://github.com/zertovitch/hac/issues

2.2 Exit codes from Shell_Execute

Sometimes, with piped commands like below, exit codes are shifted one bit right. So 0 becomes 256 and 1 becomes 0:

```
Shell_Execute ("dpkg-query -W -f='${Status}' " & Package_Name & " 2>/dev/null | grep -c 'ok installed' 1>/dev/null", Err);

if Err = 256 then
  …
```

Ada, « it's stronger than you ».
Tribute to Daniel Feneuille, legendary french Ada teacher (and much more)[10]

[10] http://d.feneuille.free.fr