### ADA 2022 Reference Card

#### Attributes

<table>
<thead>
<tr>
<th>Access</th>
<th>(\text{Access}) returns (\text{access_type})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>(\text{Address}) returns (\text{System_Address})</td>
</tr>
<tr>
<td>Adjacent</td>
<td>(\text{Adjacent}) returns (T)</td>
</tr>
<tr>
<td>Aft</td>
<td>(\text{Aft}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Alignment</td>
<td>(\text{Alignment}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Base</td>
<td>(\text{Base}) returns (\text{S})</td>
</tr>
<tr>
<td>Bit Order</td>
<td>(\text{Bit_Order}) returns (\text{System_Bit_Order})</td>
</tr>
<tr>
<td>Body Version</td>
<td>(\text{Body_Version}) returns (\text{String})</td>
</tr>
<tr>
<td>Callable</td>
<td>(\text{T_Callable}) returns (\text{Booleans})</td>
</tr>
<tr>
<td>Caller</td>
<td>(\text{T_Caller}) returns (\text{Task_ID})</td>
</tr>
<tr>
<td>Ceiling</td>
<td>(\text{T_Ceiling}) returns (T)</td>
</tr>
<tr>
<td>Class</td>
<td>(\text{Class}) returns (\text{class_wide_type})</td>
</tr>
<tr>
<td>Class</td>
<td>(\text{Class_class_wide_type}) returns (\text{class_wide_type} S)</td>
</tr>
<tr>
<td>Component Size</td>
<td>(\text{Component_Size}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Compose</td>
<td>(\text{Compose}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Constrained</td>
<td>(\text{Constrained}) returns (\text{Booleans})</td>
</tr>
<tr>
<td>Copy Sign</td>
<td>(\text{Copy_Sign}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Count</td>
<td>(\text{Count}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Definite</td>
<td>(\text{Definite}) returns (\text{Booleans})</td>
</tr>
<tr>
<td>Delta</td>
<td>(\text{Delta}) returns (\text{universal_real})</td>
</tr>
<tr>
<td>Denorm</td>
<td>(\text{Denorm}) returns (\text{Booleans})</td>
</tr>
<tr>
<td>Digits</td>
<td>(\text{Digits}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Digits</td>
<td>(\text{Digits}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Enum Rep</td>
<td>(\text{Enum_Rep}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Enum Val</td>
<td>(\text{Enum_Val}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>Exponent</td>
<td>(\text{Exponent}) returns (\text{universal_integer})</td>
</tr>
<tr>
<td>External Tag</td>
<td>(\text{External_Tag}) returns (\text{String})</td>
</tr>
<tr>
<td>First</td>
<td>(\text{First}) returns (\text{index_type})</td>
</tr>
</tbody>
</table>

**First**
- Lower bound of N-th index of [constrained] array type.

**First**
- Lower bound of first index of [constrained] array type.

**First**
- Lower bound of the range of scalar subtype.

**First**
- Bit offset, from the start of the first of the storage elements occupied by C, of the first bit occupied by C.

**First Valid**
- Denotes the smallest value that belongs to S and satisfies the predicates of S.

**Floor**
- Largest integral value less than or equal to the argument.

**Fore**
- Number of characters needed before the decimal point.

**Fraction**
- Decompose floating point argument into fractional part.

**S'Adjacent (X,Towards:T) return T**
- Adjacent floating point number to X in the direction of Towards.

**S'Alignment return universal_integer**
- Alignment of object.

**S'Compose (Fraction:T;Exponent:universal_integer) return T**
- Combine fraction and exponent into a floating point subtype.

**S'Copy_Sign (Value,Sign:T) return T**
- Result whose magnitude is that of float Value and whose sign is that of Sign.

**S'Ceiling (X:T) return T**
- Smallest (most negative) integral value greater than or equal to argument.

**S'Delta return universal_real**
- Smallest (most negative) value of floating point exponent.

**S'Denorm return Boolean**
- True if numeric overflow detected for fixed or floating point.

**S'Decimal_Mantissa return universal_integer**
- Number of digits in machine representation of mantissa.

**S'Delete return Boolean**
- True if rounding is performed on inexact results of the fixed or floating point.

**S'Digits return universal_integer**
- Number of digits of the decimal fixed point subtype.

**S'Digits return universal_integer**
- Number of decimal mantissa digits for floating point subtype.

**S'Denorm return Boolean**
- True if every value is expressible in canonical form with an exponent of \(\text{T\_Machine\_Emin}\).

**S'Digits return universal_integer**
- Number of decimal mantissa digits for floating point subtype.

**S'Digits return universal_integer**
- Number of decimal mantissa digits for floating point subtype.

**S'Compose (Fraction:T;Exponent:universal_integer) return T**
- Combine fraction and exponent into a floating point subtype.

**S'Copy_Sign (Value,Sign:T) return T**
- Result whose magnitude is that of float Value and whose sign is that of Sign.

**S'Alignment return universal_integer**
- Alignment of object.

**S'Adjacent (X,Towards:T) return T**
- Adjacent floating point number to X in the direction of Towards.

**S'Alignment return universal_integer**
- Alignment of object.

**S'Compose (Fraction:T;Exponent:universal_integer) return T**
- Combine fraction and exponent into a floating point subtype.

**S'Copy_Sign (Value,Sign:T) return T**
- Result whose magnitude is that of float Value and whose sign is that of Sign.

**S'Ceiling (X:T) return T**
- Smallest (most negative) integral value greater than or equal to argument.

**S'Delta return universal_real**
- Smallest (most negative) value of floating point exponent.

**S'Denorm return Boolean**
- True if numeric overflow detected for fixed or floating point.

**S'Decimal_Mantissa return universal_integer**
- Number of digits in machine representation of mantissa.

**S'Delete return Boolean**
- True if rounding is performed on inexact results of the fixed or floating point.
Version
P'Structure' := Structure | Structure
Version returns string that identifies the version of the compilation unit that contains the declaration of the program unit.

Wide_Image
X'String' := String | String
ns
Wide_Image returns the value of X as a String.

Wide_Value
S'Value (X:Type) := Value
S'Value returns a value given an image of the value as a Type argument (X).

Wide_Width
X'Width := universal_integer
Maximum length of X as a String.

Write
S'Write (Stream:access Ada.Streams.Root_Stream_Type'Class;X:T)
S'Write (Stream:access Ada.Streams.Root_Stream_Type'Class;X:T)
Writes X to Stream.

Address
X'I with Address => System.Address
Address of the first of the storage elements allocated.

Aggregate
S with Aggregate => (Aggregate)
Mechanism to define user-defined aggregates.

Alignment
X'I with Alignment => universal_integer
Alignment of object or subtype.

All_Call_Remote
P with All_Call_Remote => Boolean
All indirect or dispatching remote subprogram calls, and all direct remote subprogram calls, should use the Partition Communication Subsystem.

Allows Exit
P with Allows_Exit => Boolean
An indication of whether a subprogram will operate correctly for arbitrary transfers of control.

Asynchronous
P with Asynchronous => Boolean
Remote procedure calls are asynchronous; the caller continues without waiting for the call to return.

Atomic
S'I with Atomic => Boolean
Declare that a type, object, or component is atomic.

Atomic_Components
A'I with Atomic_Components => Boolean
Declare that the components of an array type or object are atomic.

Attach_Handler
P with Attach_Handler => Ada.Interrupts.Interrupt_Id
Protected procedure is attached to an interrupt.

Bit_Order
S with Bit_Order => System.Bit_Order
Order of bit numbering in a record_representation_clause.

Component_Size
A'I with Component_Size => universal_integer
Size in bits of a component of an array type.

Constant_Importing
S'I with Constant_Importing => P
Defines function to implement user-defined indexed components.

Convention
S'Name => convention_identifier
Calling convention or other convention used for interfacing to other languages.

CPU
T with CPU => System.Multiprocessors.CPU_Range
Processor on which a given task, or calling task for a protected operation, should run.

Default_Component_Value
S'I with Default_Component_Value => Component_Type
Default value for the components of an array-of-scalar subtype.

Default_Initial_Condition
S'I with Default_Initial_Condition => Boolean
If the Default_Initial_Condition aspect is specified for a type T, then the default initial condition expression applies to S and to all descendants of S.

Default_Initializing
S'I with Default_Initializing => P
Default iterator to be used in for loops.

Default_Value
S'I with Default_Value => P
Default value for a scalar subtype.

Discard_Names
S'I with Discard_Names => Boolean
Requests a reduction in storage.

Dispatching
P'I with Dispatching => dispatching_operation_specifier

Dispatching_Domain
T'I with Dispatching_Domain => System.Multiprocessors.During_Domain
Dispatching_Domain in Dispatching_Domain in Domain (group of processors) on which a given task should run.

Dynamic_Predicate
S'I with Dynamic_Predicate => Boolean
Condition that will hold true for objects of a given subtype; the subtype is not static.

Elaborate_Body
S'I with Elaborate_Body => Boolean
A given package will have a body, and that body is elaborated immediately after the declaration.

Exclusive_Functions
S'I with Exclusive_Functions => Boolean
Specifies mutual exclusion behavior of protected functions in a protected type.

Export
P'I with Export => Boolean
Entity is exported to another language.

External_Name
P'I with External_Name => String
Name used to identify an imported or exported entity.

External_Tag
S'I with External_Tag => String
Unique identifier for a tagged type in streams.

Full_Access_Only
X'I with Full_Access_Only => Boolean
Declare that a volatile type, object, or component is full access.

Global
D'I with Global => global_aspect_definition
Global object usage contract.

Global_Class
D'I with Global_Class => global_aspect_definition
Global object usage contract inherited on derivation.

Implicit_Dereference
A'I with Implicit_Dereference => Discriminant
Mechanism for user-defined implicit all.

Import
P'I with Import => Boolean
Entity is imported from another language.

Independent
S'I with Independent => Boolean
Declare that a type, object, or component is independently addressable.

Independent_Components
A'I with Independent_Components => Boolean
Declare that the components of an array type or object are independently addressable.

Input
S'I with Input => Boolean
Function to read a value from a stream for a given type, including any bounds and discriminants.

Input_Class
S'I with Input_Class => Boolean
Function to read a value from a stream for a class-wide type associated with a given type, including any bounds and discriminants.

Integer_Literal
Integer_Literal

Interrupt_Handler
Interrupt_Handler
Protected procedure may be attached to interrupts.

Interrupt_Priority
Interrupt_Priority
Priority of a task object or type, or priority of a protected object or type; the priority is in the interrupt range.
Relative_Deadline

Relative_Deadline

Real_Literal

Real_Literal

Read'Class

Read'Class

Put_Image

D with Pure

Priority

Priority

Pure

Put_Image

Read

Read

Read'Class

Real_Literal

Relative_Deadline

Relative_Deadline

Real_Literal

Real_Literal

Read

Read

Real_Literal

Real_Literal

T with Relative_Deadline => RD

Ensures that the absolute deadline of the task when created is RD of type

Real_Time.Time_Span.

Remote_Call_Interface

Remote_Call_Interface

Remote_Types

Remote_Types

Shared_Passive

Shared_Passive

Size

Size

Small

Small

Stable_Properties

Stable_Properties

Stable_Properties'Class

Stable_Properties'Class

Static

Static

Static_Predicate

Static_Predicate

Storage_Pool

Storage_Pool

Storage_Size

Storage_Size

Storage_Size (task)

Stream_Size

Stream_Size

String_Literal

String_Literal

Synchronization

Synchronization

Type_Invariant

Type_Invariant

Type_Invariant'Class

Type_Invariant'Class

Unchecked_Union

Unchecked_Union

Use_Formal

Use_Formal

Variable_Indexing

Variable_Indexing

Volatile

Volatile

Volatile_Components

Volatile_Components

Write

Write

Write'Class

Write'Class

Yield

Yield

Yield

Yield

Yield

Yield
### PRAGMAS

#### Admission_Policy

**pragma Admission_Policy (policy_identifier)**

An admission policy governs the order in which competing tasks are evaluated for acquiring the execution resource associated with a protected object.

#### All_Calls_Remote

**pragma All_Calls_Remote ([library_unit_name])**

Force all calls on a remote-call-interface library unit from other library units in the same active partition to be remote.

#### Assert

**pragma assert ([Check =>] boolean_expression [, [Message =>] string_expression])**

Raises Assertion_Error exception with an optional message when the expression is false.

#### Assertion_Policy

**pragma Assertion_Policy (Check | Ignore)**

Enables or disables assertions including pre and post conditions.

#### Atomic

**pragma Atomic ([local_name])**

Is used with types and variables to specify that the code generated must read and write the type or variable from memory atomically, i.e. as a single non-interuptible operation.

#### Atomic_Components

**pragma Atomic_Components (array_local_name)**

The components of the named array or every array of the named type is to be examined and updated atomically.

#### Attach_Handler

**pragma Attach_Handler ([handler_name , expression])**

The handler procedure is attached to the specified interrupt.

#### Conflict_Check_Policy

**pragma Conflict_Check_Policy (policy_identifier [, policy_identifier])**

This subclause determines what checks are performed relating to possible concurrent conflicting actions.

#### Convention


Directs the compiler to represent a type or subprogram using a foreign language convention.

#### CPU

**pragma CPU ([System, Multiprocessors, CPU Range])**

Processor on which a given task, or calling task for a protected operation, should run.

#### Default_Storage_Pool

**pragma Default_Storage_Pool (storage_pool_indicator)**

Specifies the storage pool that will be used in the absence of an explicit specification of a storage pool or storage size for an access type.

#### Detect_Blocking

**pragma Detect_Blocking**

Raises Program_Error when a potentially blocking operation is detected that occurs during the execution of a protected operation or a parallel construct defined within a compilation unit to which the pragma applies.

#### Discard_Names

**pragma Discard_Names ([In =>] local_name)**

Reduce the memory needed to store names of Ada entities, where no operation uses those names.

#### Dispatching_Domain

**pragma Dispatching_Domain (expression)**

Domain (group of processors) on which a given task should run.

#### Elaborate

**pragma Elaborate ([library_unit_name], ...)**

Guarantees that both the spec and body of its argument will be elaborated prior to the unit with the pragma.

**pragma Elaborate_All ([library_unit_name], ...)**

Guarantees that both the spec and body of its argument will be elaborated prior to the unit with the pragma, as well as all units withed by the spec and body of the argument, recursively.

**pragma Elaborate_Body ([library_unit_name])**

Requires that the body of a unit is elaborated immediately after its spec. This restriction guarantees that no client scenario can invoke a server target before the target body has been elaborated.

#### Export

**pragma Export ([Convention =>] convention_identifier, [Entity =>] local_name[, [External_Name =>] string_expression][, [Link_Name =>] string_expression])**

Directs the compiler to make available subprograms or data objects written in Ada to foreign computer languages.

#### Generate_Deadlines

**pragma Generate_Deadlines**

Makes the deadline of a task be recomputed each time it becomes ready. The new deadline is the value of Real_Time.Clock at the time the task is added to a ready queue plus the value returned by Get_Relative_Deadline.

#### Import

**pragma Import ([Convention =>] convention_identifier, [Entity =>] local_name[, [External_Name =>] string_expression][, [Link_Name =>] string_expression])**

Directs the compiler to use code or data objects written in a foreign computer language.

#### Independent

**pragma Independent Component ([component_local_name])**

Declare that a type, object, or component is independently addressable.

#### Independent_Components

**pragma Independent_Components ([local_name])**

Declare that the components of an array or record type, or an array object, are independently addressable.

#### Inline

**pragma Inline ([entity_name , ...])**

Directs the compiler to inline the code of the given subprogram, making execution faster by eliminating overhead of the subprogram call.

#### Inspection_Point

**pragma Inspection_Point ([object_name , ...])**

Directs the compiler to ensure that the specified variable is available where the pragma appears. This pragma aids in debugging.

#### Interrupt_Handler

**pragma Interrupt_Handler ([handler_name])**

Tell the compiler this is an interrupt handler.

#### Interrupt_Priority

**pragma Interrupt_Priority ([expression])**

Assigns the given priority to the whole protected object. No other interrupts at or below that level will be enabled whenever the procedure is executing.

#### Linker_Options

**pragma Linker_Options (string_expression)**

Used to specify the system linker parameters needed when a given compilation unit is included in a partition.

#### List

**pragma List ([identifier])**

Specifies that listing of the compilation is to be continued (On) or suspended (Off) until a List pragma with the opposite argument is given within the same compilation.

#### Locking_Policy

**pragma Locking_Policy (policy_identifier)**

Chooses locking policy.

#### No_Return

**pragma No_Return ([subprogram_local_name, subprogram_local_name])**

States that a procedure will never return normally; that is, it will raise an exception, loop endlessly, or terminate the program.

#### Normalize_Scalars

**pragma Normalize_Scalars**

Directs the compiler to initialize otherwise uninitialized scalar variables with predictable values. If possible, the compiler will choose out-of-range values.

#### Optimize

**pragma Optimize ([identifier])**

Gives advice to the implementation as to whether time (Time) or space (Space) is the primary optimization criterion, or that optional optimizations should be turned off (Off).

#### Pack

**pragma Pack (first_subtype_local_name)**

Directs the compiler to use type representations that favor conservation of storage space, rather than ease of access.

#### Page

**pragma Page**

Specifies that the program text which follows the pragma should start on a new page (if the compiler is currently producing a listing).

#### Partition_Elaboration_Policy

**pragma Partition_Elaboration_Policy (policy_identifier)**

Specifies the elaboration policy for a partition.

#### Preelaborable_Initilization

**pragma Preelaborable_Initilization (direct_name)**

Specifies that all objects of the type have preelaborable initialization expressions.

#### Preelaborate

**pragma Preelaborate ([library_unit_name])**

Slightly less restrictive than pragma Pure, but still strong enough to prevent access before elaboration problems within a unit.

#### Priority

**pragma Priority (Integer)**

Sets a task's priority. The pragma must be called in the task specification.

#### Priority_Specific_Dispatching

**pragma Priority_Specific_Dispatching (policy_identifier, first_priority_expression, last_priority_expression)**

Specifies the task dispatching policy for the specified range of priorities.
Profile

pragma Profile (profile_identifier, profile pragma argument association)
Expresses the user's intent to abide by a set of Restrictions or other specified run-time policies. These may facilitate the construction of simpler run-time environments.

Pure

pragma Pure (library unit name)
Guarantees that no scenario within the unit can result in an access before elaboration problem.

Queuing_Policy

pragma Queuing_Policy (FIFO Queuing|Priority Queuing)
Defines the queuing policy used on task entry to an Ada partition.

Relative_Deadline

pragma Relative_Deadline (Real_Time.Time_Span)
Defines deadline.

Remote_Call_Interface

pragma Remote_Call_Interface (library unit name)
Categorizes a library-unit as a Remote-Call-Interface.

Remote_Types

pragma Remote_Types (library unit name)
Categorizes a library-unit as a Remote-Type.

Restrictions

pragma Restrictions (restriction, ...) Used to forbid the utilization of some language features.

Reviewable

pragma Reviewable
Directs the compiler to provide information that aids inspection of the program’s object code.

Shared_Passive

pragma Shared_Passive (library unit name)
Allows the use of passive partitions in the context described in the Ada Reference Manual; i.e., for communication between separate partitions of a distributed application using the features in Annex E.

Storage_Size

pragma Storage_Size (expression)
Specifies the amount of space to be allocated for the task stack. This cannot be extended, and if the stack is exhausted, then Storage_Error will be raised (if stack checking is enabled).

Suppress

pragma Suppress (identifier)
Gives the compiler permission to omit checks, but does not require the compiler to omit checks.

Task_Dispatching_Policy

pragma Task_Dispatching_Policy (policy identifier)
Chooses scheduling policies.

Unchecked_Union

pragma unchecked Union (first subtype local name)
Denotes an unconstrained discriminated record subtype having a variant part.

Unsuppress

pragma unsuppress (identifier)
Unsuppresses a given check.

Volatile

pragma Volatile (local name)
Is used with types and variables to specify that the variable in question may suddenly change in value. For example, this may occur due to a device writing to a shared buffer.

Volatile_Components

pragma Volatile_Components (array local name)
notDeclares that the components of the array type — but not the array type itself — are volatile.

S TANDARD L IBRARY

package Standard

Boolean True or False
Integer Implementation defined
Natural Integers >= 0
Positive Integers > 0
Float Implementation defined
Character 8-bit ASCII/ISO 8859-1
Wide_Character 16-bit ISO 10646
Wide_Wide_Character 32-bit ISO 10646:2020
String Array of Characters
Wide_Wide_String Array of Wide_Character
Duration Time in seconds
Constraint_Error Predefined exception
Program_Error Predefined exception
Storage_Error Predefined exception
Tasking_Error Predefined exception

package Ada

Assertions
Asynchronous_Task_Control
Calendar
Arithmetic
Formatting
Time_Zones
Characters
Conversions
Handling
Latin_1
Command_Line
Complex_Text_IO
Containers
Bounded_Doubly_Linked_Lists
Bounded_Hashed_Maps
Bounded_Hashed_Sets
Bounded_Indefinite_Holders
Bounded_Multway_Trees
Bounded_Ordered_Maps
Bounded_Ordered_Sets
Bounded_Priority_Queue
Bounded_Synchronized_Queue
Bounded_Vectors
Doubly_Linked_Lists
Generic_Array_Sort
Generic_Constrained_Array_Sort
Generic_Sort
Hashed_Maps
Hashed_Sets
Indefinite_Doubly_Linked_Lists
Indefinite_Hashed_Maps
Indefinite_Holders
Indefinite_Multiway_Trees
Indefinite_Ordered_Maps
Indefinite_Ordered_Sets
Indefinite_Vectors
Multway_Trees
Ordered_Maps
Ordered_Sets
Synchronized_Queue_Interfaces
Unbounded_Priority_Queue
Unbounded_Synchronized_Queue
Vectors
Decimal
Direct_IO
Directories
Hierarchical_File_Names
Information
Dispatching
EDF
Non_Preemptive
Round_Robin
Dynamic_Priorities
Environment_Variables
Exceptions
Execution_Time
Group_Budgets
Interrupts
Timers
Finalization
Float_Text_IO
Float_Wide_Text_IO
Float_Wide_Wide_Text_IO
Integer_Text_IO
Integer_Wide_Text_IO
Integer_Wide_Wide_Text_IO
Interrupts
Names
IO_Exceptions
Iterator_Interfaces
Locales
Numerics
Big_Numbers
Big_Integers
Big_Reals
Complex_Arrays
Complex_Elementary_Functions
Complex_Types
Discrete_Random
package Interfaces
C
Pointers
Strings
COBOL
Fortran

package System
Address_To_Access_Conversions
Atomic_Operations
Exchange
Integer_Arithmetic
Modular_Arithmetic
Test_And_Set
Machine_Code
Multiprocessors
Dispatching_Domains
RPC
Storage_Elements
Storage_Pools
Subpools