Ada Cheat Sheet

|  |  |
| --- | --- |
| Types |  |
| Typedef  Predefined types  get size of type (bits)  Enumeration types  Integer types   Unsigned / Modular types  Floating Point Types  Fixed Point Types  Composite Types  Record / Struct  Subtypes | type TYPENAME is VALUE;  Integer, Float, Boolean, Character, String,   TYPENAME’size -> example: Integers’size  Example Boolean -> true, false type Boolean is  (true, false);  signed: Integer own: type My\_Int is range 1..100; 1-maxInteger: Positive 0-maxInteger: Natural   type byte is mod 2\*\*8;  type ex\_values is digits 10 range -1.0..1.0;  type ordinary\_dist is delta 0.001 range 0.0..1.0; -> 2^-10  type decimal\_dist is delta 0.01 digits 9 range 0.0..9\_999\_999.99;  type Own\_String is array (1..10) of Integer; type String is array(Positive range <>) of Character;  Ordinary (not extendable through inheritance): type Inventory\_Item is record  UPC\_Code : String(1..20); end record;  Tagged (extendable through inheritance):  type Person is tagged record  Name : String(1..20);  end record; type Employee is new Person with record  Id : Integer;  end record;  subtype Rainbow is Color range Red .. Blue; |
| Ranges |  |
| For scalar types  Subtypes  Loops  First Last Range | type Rankings is new Integer range 1..10;   * see subtypes   for Num in 1..10 loop  …. end loop;  Days’First  Days’Last Voltages’Range == Voltages’First..Voltages’Last |
| Operators |  |
| Assignment Equality NonEquality  Modulus Remainder  AbsoluteValue Exponentiation  Membership Log AND == Bit AND  String Concatination | :=  =  /=  mod  rem  abs  \*\*  In  and (same: or, xor, not)  & |
| Constructor / Destructor like blocks |  |
| Constructor with function  Advanced using Initialize and Finalize | type T is tagged record  F : Integer := init\_function;  end record;  function init\_function return Integer is  begin  Put\_Line ("Compute");  return 0;  end init\_function;  V1 : T;  V2 : T := (F => 0);  type T is new Ada.Finalization.Controlled with record  F : Integer;  end record;  procedure Initialize (Self : in out T) is  begin  Put\_Line ("Compute");  Self.F := 0;  end Initialize;  V1 : T;  V2 : T := (F => 0); |
| Loops |  |
| Loop  While  for | loop  if condition then  exit;  end if;  end loop;  while condition loop  …  end loop  for var in low\_value .. high\_value loop  …  end loop; |
| Conditions |  |
| If  Switch case | If condition then  …  end if;  case expression is  when choice =>  ….  when choice2 =>  ….  end case; |
| Subprograms |  |
| Procedure (no return value)  Function (always return value) | procedure function\_name(in1, in2 : IN OUT Integer) is  Temp : Integer := Left;  begin  Right := Temp;  end function\_name;  IN OUT -> initial value and expected to be written to  IN -> Read Only constant  OUT -> No initial value but expected to be written to    Only IN parameter |
| Package handling |  |
| define package  use package | package PACKAGENAME is end PACKAGENAME;  with PACKAGENAME;  use PACKAGENAME; |
| Concurrency   protected type  task |  |
|  |  |
| Visibility / inheritance |  |
|  |  |
| Generics / Templates |  |
|  |  |
| Useful Building Blocks |  |
| Std. Output  Std. Input  File IO  Create file  Write single to file  Set output to file  Close file  Open file  Read char  Read line  Reset position in file  Skip line | Package Ada.Text\_IO / Ada.Integer\_Text\_IO   * Put(OUTPUT) -> single character * Put\_Line(OUTPUT) ->line   \*\*   * Get(s) -> reads s.length input to s (ignores new lines) * Get\_Line(s, len) -> reads len length input to s * Filevar : FILE\_TYPE; Create(Filevar, Out\_File, “filename.txt”); * Put(Filevar, “output text”) * Set\_Output(Filevar); Put(“output text”); Put\_Line(“output line”); New\_Line(n); -> n = number of new lines Set\_Output(Standard\_Output); * Close(Filevar) * Open(Filevar, In\_File, “filename.txt”) * Get(Filevar, c) -> c = input char * loop  exit when End\_Of\_File(Filevar);  Get(Filevar, c);  If End\_Of\_Line(Filevar) then  …  else  Put(c);   end if  end loop;   * Reset(Filevar); * Skip\_line; |
|  |  |