* developed by the United States Department of Defense in an attempt to solve the software muddle as it existed in the mid 1970's
* Ada 95 is an ISO update to the Ada programming language to incorporate the latest knowledge of software development into the language
* Errors are reported at compile time rather than execution.
* **Identifier** - any object in Ada.
  + Must start with a letter.
  + Made up of as many letters, numbers, and underlines as desired provided that the underlines occur only singly, and an underline is not the last character.
  + Case of letters is not significant.
  + There is no limit to the length of an identifier but each identifier must fit on one line of text and the writer of the compiler may impose a line length limit. The minimum line length must be at least 200 characters.
  + No blanks or special characters can be used as part of an identifier.
* Ada 95 uses 69 identifiers which are called reserved words. Note that Ada 83, by contrast, only had 63 reserved words.
  + Reserved words are reserved for specific uses within an Ada program and cannot be used for any other purpose.
  + Lower case
* Variables - All variables will be written with the initial letter of each word capitalized, and all others in lower case.
* TYPES - All types will be written in all capital letters.
* CONSTANTS - All constants will be written in all capital letters.
* ENUM VALUES - All enumerated values will be written in all capital letters.
* ATTRIBUTES - All attributes will be written in all capital letters.
* Procedure Names - All procedure names will be written with the initial letter of each word capitalized and all others in lower case.
* Function Names - Same as procedure names.
* Package Names - Same as procedure names.
* Library Names - Same as procedure names.
* Renaming - Ada gives you, the programmer, the ability to assign a new name to various entities in a program for your own convenience. Ada permits the renaming of objects, exceptions, task entries, and subprograms. It is simply an alias which can be used to refer to the entity which is renamed.
* Ada allows overloading.

**procedure** <program name> **is**

       <declarative part>

**begin**

       <executable part>

**end** <optional repeat of program name>;

* Put – print output
* New\_line – print new line
* Put\_line – print output with automatic carriage return

with Ada.Text\_IO, Ada.Integer\_Text\_IO;

use Ada.Text\_IO, Ada.Integer\_Text\_IO;

procedure OneInt is

Index : INTEGER; -- A simple Integer type

begin

Index := 23;

Put("The value of Index is");

Put(Index); -- The default field width is 11 columns

New\_Line;

Index := Index + 12;

Put("The value of Index is");

Put(Index, 8);

New\_Line;

end OneInt;

* type <type\_name> is <type\_definition>;

subtype <subtype\_name> is <subtype\_definition>;

EXAMPLE: type MY\_INT\_TYPE is range -10\_000..20\_000;

* Example of boolean:

if Index = 12

and if Count = 12

and if Truth currently has the value TRUE

and if TRUE (which is always TRUE)

then assign TRUE to Question

otherwise assign FALSE to Question.

* Order of precedence:

|  |  |
| --- | --- |
| \*\* not abs | Highest precedence |
| \* / mod rem | Multiplying operators |
| + - | Unary operators |
| + & | Binary adding operators |
| = /= < | Relational operators |
| <= > >= | Relational operators |
| in not in | (same precedence) |
| and or xor | Logical operators |
| and then or else | (same precedence) |

* **Loop statements:**

loop <...> end loop;

while <BOOLEAN expression> loop <...> end loop;

for <loop index> in <range> loop <...> end loop;

if <condition> then <...> end if;

case <selector> is <...> end case;

EXAMPLES:

Index := 1;

loop -- This is the simplest loop

Put("Index =");

Put(Index, 5); New\_Line;

Index := Index + 1;

exit when Index = 5;

end loop;

Index := 1;

loop -- Another simplest loop

Put("Index =");

Put(Index, 5); New\_Line;

Index := Index + 1;

if Index = 5 then exit; end if;

end loop;

Count := 1;

while Count < 5 loop -- This is the while loop

Put("Count =");

Put(Count, 5); New\_Line;

Count := Count + 1;

end loop;

for Index in 1..4 loop -- This is the for loop

Put("Doubled index =");

Put(2 \* Index, 5); New\_Line;

end loop;

for Count in reverse 5..8 loop -- This is the reverse for loop

Put("Triple count =");

Put(3 \* Count, 5); New\_Line;

end loop;

for Index in 7..11 loop -- An empty loop

null;

end loop;

* Float types: SHORT\_FLOAT, LONG\_FLOAT
* Integer types: SHORT\_INTEGER, LONG\_INTEGER
* **User defined type examples:**

type <type-name> is new <existing-type>;

type <type-name> is range <lower-limit>..<upper-limit>;

* **Multiple procedure example:**

procedure Proced2 is

Counter : INTEGER;

procedure Write\_A\_Header is

begin

Counter := 1;

Put("This is the heading for this little program.");

New\_Line(2);

end Write\_A\_Header;

procedure Write\_And\_Increment is

begin

Put("This is line number");

Put(Counter, 2);

Put\_Line(" of this program.");

Counter := Counter + 1;

end Write\_And\_Increment;

procedure Write\_An\_Ending\_Statement is

begin

New\_Line;

Put\_Line("This is the end of this little program.");

end Write\_An\_Ending\_Statement;

begin

Write\_A\_Header;

for Index in 1..7 loop

Write\_And\_Increment;

end loop;

Write\_An\_Ending\_Statement;

end Proced2;

* **Array examples:**

type MATRIX is array(INTEGER range 1..3,🡨(rows, columns)

INTEGER range 1..4) of INTEGER;

* Iterating through the loop:

for Across in 1..3 loop

for Over in 1..4 loop

Square\_Board(Across, Over) := Across \* Over;

Chess\_Board(Across, Over) := 0;

end loop;

end loop;

* Assigning a value:

Checker\_Board(2, 3) := 2;

Checker\_Board(Checker\_Board(2, 3), 4) := 17;

Checker\_Board(3, 3) := Chess\_Board(3, 3);

* **String/Characters**:
  + STRING
  + WIDE\_STRING
  + CHARACTER
  + WIDE\_CHARACTER
  + WIDE used for international languages
  + Ada.Characters.Handling
    - Is\_Upper
    - Is\_Lower
* **Record:**

type DATE is

record

Month : INTEGER range 1..12;

Day : INTEGER range 1..31;

Year : INTEGER range 1776..2010;

end record;

Independence\_Day : DATE;

Independence\_Day.Month := 7;

Independence\_Day.Day := 4;

Independence\_Day.Year := 1776;

* **Writing to a text file:**

-- First we create the file

Create(Turkey, Out\_File, "TEST.TXT");

-- Then we write to it

Put\_Line(Turkey, "This is a test of turkey");

Put(Turkey, "and it should work well.");

New\_Line(Turkey, 2);

Put\_Line("Half of the turkey test");

Set\_Output(Turkey); -- Make Turkey the default output

Put\_Line("This is another test of turkey");

Put("and it should work well.");

New\_Line(2);

Put\_Line(Standard\_Output, "Half of the turkey test");

Set\_Output(Standard\_Output); -- Return to the Standard default

Put\_Line("Back to the standard default output.");

Close(Turkey);

* **Reading from a text file:**

My\_File : FILE\_TYPE;

open(My\_File, In\_File, "CHARACTS.TXT");

loop -- Read one character at a time and display it

exit when End\_Of\_File(My\_File);

Get(My\_File, One\_Char);

Put(One\_Char);

end loop;

New\_Line(2);

Reset(My\_File); -- Reset and start over with the same file

loop -- Read and display but search for End of lines

exit when End\_Of\_File(My\_File);

Get(My\_File, One\_Char);

if End\_Of\_Line(My\_File) then

Put("<--- End of line found");

New\_Line;

else

Put(One\_Char);

end if;

end loop;

New\_Line;

Reset(My\_File); -- Reset and start over the third time

-- Read and display but search for End of lines

loop -- using a look ahead method

exit when End\_Of\_File(My\_File);

Get(My\_File, One\_Char);

Put(One\_Char);

if End\_Of\_Line(My\_File) then

Put("<--- End of line found");

New\_Line;

end if;

end loop;

Close(My\_File);

* **Packages:**
  + **Create a package:**

-- Interface of AdderPkg

package AdderPkg is

type MY\_ARRAY is array(INTEGER range <>) of FLOAT;

procedure Add\_Em\_Up(In\_Dat : in MY\_ARRAY;

Sum : out FLOAT);

end AdderPkg;

-- Implementation of AdderPkg

package body AdderPkg is

procedure Add\_Em\_Up(In\_Dat : in MY\_ARRAY;

Sum : out FLOAT) is

Total : FLOAT;

begin

Total := 0.0;

for Index in In\_Dat'FIRST..In\_Dat'LAST loop

Total := Total + In\_Dat(Index);

end loop;

Sum := Total;

end Add\_Em\_Up;

end AdderPkg;

* + **Use the package:**

with AdderPkg;

use AdderPkg;

* **Inline pragma:**
  + Tells the compiler to expand the called subprogram and insert it into the calling program for each call
  + pragma INLINE(subprogram1, subprogram2, ... );
* **Enumerator array example:**

type DAY is (MON, TUE, WED, THU, FRI, SAT, SUN);

Hours : array(DAY) of FLOAT;

Total\_Hours : FLOAT;

Today : DAY;

for Today in MON..FRI loop

Hours(Today) := 8.0;

end loop;

* **Private example:**

type DATA\_STRUCTURE is private;

…

private

type DATA\_STRUCTURE is

record

Value1 : INTEGER;

Value2 : INTEGER;

Value3 : INTEGER;

end record;

* **Inheritance example:**

package Starter is

type MY\_INTEGER is new INTEGER range 5..150;

type MY\_FLOAT is new FLOAT;

type WOODEN\_BOX is

record

Length : FLOAT;

Width : FLOAT;

Height : FLOAT;

end record;

function "+" (Left, Right : WOODEN\_BOX) return WOODEN\_BOX;

type STEEL\_BOX is new WOODEN\_BOX;

function "-" (Left, Right : STEEL\_BOX) return STEEL\_BOX;

end Starter;

* **Tasking example:**

with Ada.Text\_IO, Ada.Integer\_Text\_IO;

use Ada.Text\_IO, Ada.Integer\_Text\_IO;

procedure Task1 is

task First\_Task;

task body First\_Task is

begin

for Index in 1..4 loop

Put("This is in First\_Task, pass number ");

Put(Index, 3);

New\_Line;

end loop;

end First\_Task;

task Second\_Task;

task body Second\_Task is

begin

for Index in 1..7 loop

Put("This is in Second\_Task, pass number");

Put(Index, 3);

New\_Line;

end loop;

end Second\_Task;

task Third\_Task;

task body Third\_Task is

begin

for Index in 1..5 loop

Put("This is in Third\_Task, pass number ");

Put(Index, 3);

New\_Line;

end loop;

end Third\_Task;

begin

Put\_Line("This is in the main program.");

end Task1;

-- Result of Execution

-- This is in Third\_Task, pass number 1

-- This is in Third\_Task, pass number 2

-- This is in Third\_Task, pass number 3

-- This is in Third\_Task, pass number 4

-- This is in Third\_Task, pass number 5

-- This is in Second\_Task, pass number 1

-- This is in Second\_Task, pass number 2

-- This is in Second\_Task, pass number 3

-- This is in Second\_Task, pass number 4

-- This is in Second\_Task, pass number 5

-- This is in Second\_Task, pass number 6

-- This is in Second\_Task, pass number 7

-- This is in First Task, pass number 1

-- This is in First Task, pass number 2

-- This is in First Task, pass number 3

-- This is in First Task, pass number 4

-- This is in the main program.

* **Task with entry/accept example:**

with Ada.Text\_IO;

use Ada.Text\_IO;

procedure HotDog is

task Gourmet is

entry Make\_A\_Hot\_Dog;

end Gourmet;

task body Gourmet is

begin

Put\_Line("I am ready to make a hot dog for you");

for Index in 1..4 loop

accept Make\_A\_Hot\_Dog do

delay 0.8;

Put("Put hot dog in bun ");

Put\_Line("and add mustard");

end Make\_A\_Hot\_Dog;

end loop;

Put\_Line("I am out of hot dogs");

end Gourmet;

begin

for Index in 1..4 loop

Gourmet.Make\_A\_Hot\_Dog;

delay 0.1;

Put\_Line("Eat the resulting hot dog");

New\_Line;

end loop;

Put\_Line("I am not hungry any longer");

end HotDog;

-- Result of execution

-- I am ready to make a hot dog for you

-- Put hot dog in bun and add mustard

-- Eat the resulting hot dog

--

-- Put hot dog in bun and add mustard

-- Eat the resulting hot dog

--

-- Put hot dog in bun and add mustard

-- Eat the resulting hot dog

--

-- Put hot dog in bun and add mustard

-- I am out of hot dogs

-- Eat the resulting hot dog

--

-- I am not hungry any longer

* **Select (rendezvous) statement:**

select

<entry call>;

or

>entry call>;

else

<entry call>;

end select;

* **Delayed rendezvous:**

select

delay <time>;

<entry call>;

or

delay <time>;

<entry call>;

or

delay <time>;

<entry call>;

end select;

* **Task type example:**

with Ada.Text\_IO;

use Ada.Text\_IO;

procedure TaskType is

task type SHORT\_LINE is

end SHORT\_LINE;

task type LONG\_LINE is

end LONG\_LINE;

Cow, Dog, Pig : SHORT\_LINE;

Elephant, Hippopotamus : LONG\_LINE;

task body SHORT\_LINE is

begin

for Index in 1..4 loop

delay 0.0;

Put\_Line("This is a short line");

end loop;

end SHORT\_LINE;

task body LONG\_LINE is

begin

for Index in 1..3 loop

delay 0.0;

Put\_Line("This is a much longer line to be displayed");

end loop;

end LONG\_LINE;

begin

Put\_Line("This is an example of use of a task type");

end TaskType;

* **Tasking priorities**:

task type SHORT\_LINE is

pragma PRIORITY (5);

end SHORT\_LINE;

task type LONG\_LINE is

pragma PRIORITY (1);

end LONG\_LINE;