Ada Background

- Developed by DOD to replace more than 450 other languages.

- Ada was the Countess of Lovelace and daughter of the famous daughter of the famous poet, Lord Byron.

- She was a mathematician in early 1800s

- Often given the honor of being considered as the first computer “programmer”.

- Wrote programs for Charles Babbage’s Analytical Engine.

- Both were about 100 years ahead of their time.

- Published as a standard in February 1983: ANSI/MIL-STD-1815A.
Ada Literals

- **Integer Literals**
  
  → Can be written using underscores to increase readability/writability

  → 5737639019 can also be written as 5_737_639_019

- **Real Literals**
  
  → Must have a digit on each side of the decimal point

  → Also allow underscores (but not around the decimal)

  → Support for scientific notation, the literal 1.2345E3 is legal

  → Support for real binary literals, for example 2#0.101#
Ada Comments

● Comments

→ Start with a "- -" and continue until end of line:

-- comments may take up a full line
X := 2#1010#; -- may be placed after code
Y := -- placement of this comment is okay
100;

● Control Structures

if Boolean Expression then
  statements
end if;
Ada Control Structures

- if then else statement:

  if Boolean Expression then
    statements
  else
    statements
  end if;

- if then elsif statements:

  if Boolean Expression then
    statements
  elsif
    statements
  elsif
    statements
  else
    statements
  end if;
Ada Case Statements

• Case statements:

(note no break needed)

``` ada
case Flock_Of_Birds is
  when Geese =>
    Put ("A gaggle of geese");
  when Sparrows =>
    Put ("A host of sparrows");
    Put ("any number of statements");
  when others =>
    Put ("Just a flock of birds");
end case;
```

• Pipes can be used between choices in case statements:

``` ada
when choice1 | choice2 =>
  Put ("choice1 or choice2 was taken");
```
Ada Case Statements

• “..” can be used to indicate a range of values:

```ada
when 1 .. 10 =>
Put ("The choice was from 1 to 10.");
```

• Four rules to follow when using case statements in Ada:
  
  → 1. The case expression must be a discrete type.
  
  → 2. Every possible value of the case expression must be covered in one and only one when clause.
  
  → 3. If the when others clause is used, it must appear as a single choice at the end of the case statement.
  
  → 4. Choices in a when clause must be static.
Ada loops

- Loop statements:

  **Loop** - loop forever

  statements

  end loop;

  **While** Boolean expression **loop**

  statements

  -- loop until the Boolean

  -- expression becomes false

  end loop;

(here the loop counter cannot be modified in the loop)

**for** loop counter **in** discrete range **loop**

  statement

  end loop;
Ada Arrays

- Arrays

```ada
type array name is array (index specification) of type;

type Beans is (Lima, Lentil, Garbanzo, Pinto, Kidney, Black, Soy, Mung, Jelly);

type Count_Type is array (Beans range Pinto..Jelly) of Natural;

Bean_Count: Count_Type;

Bean_Count (Soy) := 181;

Bean_Count (Jelly) := -5; -- error, component value out of bounds

Bean_Count (Lentil) := 24; -- error, index out of bounds
```
Ada Arrays

● We can use array aggregates with named and positional notation

Bean_Count := ( Kidney => 5, Pinto => 4, Soy => 3, Black => 8, Mung =>10 , Jelly => 2000 );

Bean_Count := (1, 2, 3, 4, 5, 6, 7);

● We can also use the others keyword to assign a value to array components that are not explicitly given a value.

Bean_Count := (others => 0);

Bean_Count := (jelly => 2000, others => 0);
Ada Multidimensional arrays

- Multidimensional arrays

```ada
type Days_Open is (Mon, Tue, Wed, Thu, Fri);

type Daily_Count is array (Days_Open, Beans) of Natural;

Bean_Stock: Daily_Count;

Bean_Stock (Mon, Jelly) := 2000;
```

- Anonymous Arrays

```ada
Bean_StockA, Bean_StockB: array (Bean) of Natural;

(populate Bean_StockA)

Bean_StockB := Bean_StockA;  -- Illegal; type mismatch
```

Ada, Slide 10
Ada Records

```ada
type record name is
  record
    record components
  end record;

type Position is
  record
    X_Coord: Integer;
    Y_Corrd: Integer;
  end record;

● Positional and named aggregates:
  Point: Position;
  Point := (1, 2);
  Point := (X_Coord => 1, Y_Coord => 2);

● Reference and assignment :
  Point.X_Coord := Point.Y_Coord;
```
Ada Procedures

procedure procedure name (parameter definitions) is

  declarations

begin

  statements

end procedure name;

● Procedures can also have separate specification and body.

procedure procedure name
  (parameter definitions);

-- specification
Ada Procedures

```ada
procedure procedure name (parameter definitions) is

-- body
declarations
begin

  statements

end procedure name;
```

- Ada provides three different parameter modes for controlling the direction in which data can flow in and out of subprograms:
  - `in` -- allows data to flow in only one direction: from caller to subprogram
  - `out` -- allows data to flow the opposite direction: from called subprogram back to its caller.
  - `in out` -- allows data to flow in both directions.
Ada Functions

- Only the in parameter mode applies to functions.

- Functions:

  ```ada
  function function name (parameter definitions) return type is
    declarations
  begin
    statements
  end
  ```

- Ada supports overloading

- Ada supports both recursive functions and procedures

- Ada supports Inline subprogram with pragmas
Ada Subunits

- Ada supports subunits with the separate keyword:

```ada
procedure Main is

-- constants, types, and variables for Main
Flag: Boolean := True;

procedure P1 (Item: in Boolean) is separate;
  -- stub for P1

-- can still reference Flag

function F1 return Boolean is separate; -- stub for F1

-- can still reference Flag and P1
begin -- beginning of Main
  Flag := F1;
  P1 (Flag);
end Main;
```

Ada, Slide 15
Ada Subunits

```ada
separate (Main)
procedure P1 (Item: in Boolean) is
    -- constants, type, and variables for P1
begin
    -- executable statements for P1
end P1;

separate (Main)
function F1 return Boolean is
    -- constants, type, and variables for F1
begin
    -- executable statements for F1
end F1
```

- Ada also supports **with**ing subprogram library units.
Ada Packages

- Two types of packages: standard and generic
- packages consist of two parts: specification and body
- users of a package only form a dependency on the specification

```ada
-- package specification

cpackage package name is
    visible declarations
end package name;

-- package body

cpackage body package name is
    hidden declarations
begin -- initialization
    statements
end package name;
```
Ada Packages

• Following is a package that consists of only a specification:

```ada
package Houses is

  type Styles is ( Queen_Anne, Gothic
                   Craftman, Prairie_Style, English_Tudor,
                   Mediterranean);

end Houses
```

• with -- makes library units available

• use -- makes library units directly visible

• Packages may be compiled independently in which case they become library units or packages may be embedded:
Ada Embedded Packages

procedure P is

package Q is

    A: Integer := 1;

end Q

B: Integer := Q.A;

use Q;

C: Integer := A;

begin

    A := B + C;

end P;

● Ada packages support private and limited private types.

  → Private types allow you to encapsulate a types implementation. They also limit operations to assignment and equality

  → limited private types further restrict this to only implementer/user provided operations.
Ada Generics

- Generics act as a template from which actual non-generic units can be created.

```ada
-- generic specification
generic

    generic formal parameters

function function name (parameter definitions) return type is

    declarations

begin

    statements

end function name;
```

- Formal generic parameters consist of dummy names that will be replaced by actual parameters when the generic is instantiated.
Ada Generics

- Example generic formal parameter:

  ```ada
  type Item is (<>);
  ```

- Example instantiation:

  ```ada
  function Larger_Integer is new Maximum (Item => Integer);
  ```

- The (<>) can be replaced with any Discrete type at instantiation time.

- Stack Example