Ada

Presented by
Paul Fisher
New York University
pmf296@nyu.edu
Adapted from slides by Deepti Verma

Ada Installation and Example

- Follow link on NYU Classes for installation (Resources -> Ada Resources)
- Compile the examples

Introduction

- What is Ada and motivation
- Basics
- Subprograms
  - Procedures and functions
  - Packages
  - Tasks

What is Ada?

- Structured
- Statically Scoped
- Statically Typed
- Object-oriented high level programming language
- Used by U.S. Department of Defense (DoD) for real-time embedded systems, Large-scale information systems, Distributed systems, Scientific computation and safety-critical systems

Ada and Courant

- Jack Schwartz (founder of CS dept.) developed SETL (High level based on mathematical theory of sets). SETL was then used for the first valid implementation of Ada called NYU Ada/ED
- Adacore Executive Team: Robert Dewar (President Adacore, Emeritus Professor CS dept. NYU) Edmond Schonberg (Professor CS dept. NYU) Richard Kenner (Researcher CS dept. NYU) Franco Gasperoni (PhD from CS dept. NYU)

Compilation units

- An Ada program is composed of one or more units of
  - Subprograms - Procedures or Functions – define executable algorithms
  - Packages – define collection of entities
  - Tasks- computation that can occur in parallel with other computations
- Additionally there are other units like
  - Protected
  - Generic

Procedures

- A procedure call is a statement that does not return any value
- Subprogram parameters modes:
  - ‘in’ - value may be used but not changed (Default mode)
  - ‘out’ - value may be changed but not used
  - ‘in out’ - value may be changed and/or used ‘access’

  Example:

  procedure Average(A, B : in Integer; Result : out Integer)

Functions

- Unlike procedure a Function returns a value
- Parameters mode remain same as procedures
- function Average Two(A, B : in Integer) return Integer;

Packages

The package is Ada's basic unit for defining a collection of logically related entities.

Each Program unit consists of two parts :

- Declaration/Specification - contains information that will be visible to other program units. Defines the interface of the unit. Analogous to '.h' file in C
- Body - contains implementation details that need not be visible to other parts. Implementation details of the unit. Analogous to '.c' file in C

Package Specification

- File with package specification end with .ads file extension.

```
package stack is
procedure push ( x : integer ) ;
function pop return integer ;
end stack ;
```

Package Body

File with package body end with .adb file extension

```
package body stack is
procedure push ( x : integer ) is
begin
        –– Do something here
end ;
function pop return integer is
begin
        –– Do something here end
        --return value;
end stack ;
```

Tasks

- An independent execution of part of the same static code, having a stack, program counter and local environment but shared memory.
- Runs concurrently with other tasks.
- Ada tasks communicate through
  - Rendezvous - message passing
  - Shared Variables
  - Protected objects

Tasks

- Task units are similar to packages in that they have a specification and body.
  - Task declaration defines entities exported from the task (entries.)
  - Task body contains local declarations and statements of the task. A task body defines what the task will do when it is started up.

Example:

```
task taskName is
      declarations of exported identifiers
end taskName;
task body taskName is
begin
      statements
end taskName;
```

Tasks

- If there is nothing to be exported, the task specification can be simplified to:

  task taskName;

- Task Types:
  - Allows the task units to be created dynamically.

```
task type T is
        ..
end T;
Task1, Task2 : T;
```

Rendezvous

- Tasks can send messages between each other.
- Rendezvous happens using entry and accept statements
- Ada supports communication from task to task by means of the *entry call*.

Entry:

- An **entry** looks much like a procedure
- Information passes between tasks through the actual parameters of the entry call.

Rendezvous

Accept:

- An "accept" statement waits for some other task to make a request via the corresponding "entry".
- When another other task makes the matching request, the accepting task runs the accept statements between the word "do" and the "end" that matches the accept statement.