Ada Programs and SubPrograms

Industrial

Formal Methods

SWEN421-2016

Where to get answers

1. As ever Google speedily provides many excellent answers.
2. There is a link to Ada WikiBooks on the course home page that is well worth getting familiar with.
3. Us old people that like physical books can try

Building High Integrity Applications with SPARK

A simple Ada program

- Ada Packages are like java classes
- Importing classes becomes with <package>
- A simple program is in one <fname>.adb file with one computational unit -- a single procedure.
- procedure <name> is <declarations>

  begin <program>

  end;
- Declarations may be **data** or **subprograms**
- Subprograms may be procedures of functions

Variables in simple programs

- A variable is visible in the procedure in which it is declared.
- Procedures can be nested and thus variable scope has the same nested structure.
- More local variables hide the more global.
- Variables (even hidden variables) can be referenced using the dot notation.
- x := global_p.x; – x is local an hides global_p.x

Subprograms

- Functions return a value procedures do not.
- Subprograms may have parameters with modes in, out or in out. The mode indicates the flow of information.
- SPARK restricts function to in parameters
- Parameters with mode out or in out allow a procedure to have **side effects** that is they may change the state of the program.

Generic Types

- type T is limited private; any type
- type T is private; type with = and :=
- Type T is (<>); any discrete type
- Parameters of a generic formal type be instantiated with values any subtype

Arrays

- Ada only uses arrays of a fixed size
- A constrained array has its size fixed at compile time an unconstrained array has its size fixed at run time.
- Generic formal arrays and instantiating arrays must **both be unconstrained** or both be constrained.

Generic subprograms

- Subprograms with generic type parameters .
- Generic function must be:
  1. specified separately then
  2. Defined, the code body defined and finally
  3. instantiated before the function is used.

Generic functions

- Specification

```
generic
type Ix_T is private;
function get_el (i : in Ix_T) return  El_T;
```

- Implementation

```
function get_el (i : in Ix_T) return  El_T is
   begin
      return arr(i);
   end get_el;
```

- Type instantiation

```
function get_myel is get_el (Ix_T => Natural);
```

Exercise

- Define a procedure that adds elements to an array with Positive index but with size only known at run time.
- The procedure must also maintain a count that is zero or points to the last element in the array.

Example

```
procedure progs with SPARK_mode is
   generic
     type elT is private; -- assign and equal
     type arpT is array (Natural range <>) of elT;  -- unconstrained
   procedure addEl(ar: in out arpT; el: in elT; cnt: in out Natural);
```

```
   procedure addEl(ar: in out arpT; el: in elT; cnt: in out Natural) is
   begin
       if cnt < ar'Length then
          cnt := cnt +1;
         ar(cnt) := el;
         Put_Line("ping");
      end if;
   end addEl;

    type arType  is array(Natural range <>) of Integer; -- unconstrained
    procedure myaddEl is new addEl(
                        elT => Integer,
                        arpT => arType);
```

Example

```
   ardata : arType (1..10);  -- constrained
   cnt : Natural := 0;

begin
   Put("Hello ");
   myaddEl(ardata,11,cnt);
   Put_Line(" ***");

end progs;
```