Generic Examples

Generic

- Parametric polymorphism
- code reuse improves the productivity and the quality of software

```
generic
type Element_T is private; -- formal type parameter
procedure Swap (X, Y : in out Element_T);

procedure Swap (X, Y : in out Element_T) is
Temporary : Element_T := X;
begin X := Y; Y := Temporary; end Swap;

procedure Swap_Integers is new Swap (Integer);
procedure Swap_Floats is new Swap (Float);
```

Generic parameters

The generic unit declares *generic formal parameters*, which can be:

- objects (of mode *in* or *in out* but never *out*)
- types
- subprograms
- instances of another, designated, generic unit.
- When instantiating the generic, the programmer passes one *actual parameter* for each formal.
- Formal values and subprograms can have defaults, so passing an actual for them is optional.

Generic formal objects

- Formal parameters of mode *in* accept any value, constant, or variable of the designated type.
- The actual is copied into the generic instance, and behaves as a constant inside the generic;
- the designated type cannot be limited.
- It is possible to specify a default value

```
generic
Object : in Natural := 0;
```

Generic formal types

- The syntax allows the programmer to specify which type categories are acceptable as actuals
- A type declared with the syntax type T (<>) denotes a type with *unknown discriminants*

type T **is** **private**; -- Any nonlimited definite type, it is possible to assign to variables of this type and to declare objects without initial value

type T **is** (<>); -- Any discrete type: integer, modular, or enumeration.

type T **is** **range** (<>); -- Any signed integer type

type T **is** **digits** <>; --Any floating point type

type T (<>) **is** **private**; Any nonlimited type: the generic knows that it is possible to assign to variables of this type, but it is not possible to declare objects of this type without initial value.

type T (<>) **is** **limited** **private**; -- Any type at all. The actual type can be limited or not, indefinite or definite, but the *generic* treats it as limited and indefinite, i.e. does not assume that assignment is available for the type.

Generic formal subprograms

- It is possible to pass a subprogram as a parameter to a generic.
- The actual must match this parameter profile.

```
generic
type Element_T is private;
with function "*" (X, Y: Element_T) return Element_T;
function Square (X : Element_T) return Element_T;

function Square (X: Element_T) return Element_T is
begin
return X * X; -- formal operator "*".
end Square;
```

Generic formal subprograms

```
with Square; with Matrices;
procedure Matrix_Example is
function Square_Matrix is new Square
(Element_T => Matrices.Matrix_T,
 "*" => Matrices.Product);
A: Matrices.Matrix_T:=Matrices.Identity;
begin
A := Square_Matrix (A);
end Matrix_Example;
```

Generic formal subprograms

- It is possible to specify a default with "the box” *is <>;*

```
generic
type Element_T is private;
with function "*" (X, Y: Element_T)
return Element_T is <>;
```

- at the point of instantiation, a function "*" exists for the actual type, and if it is directly visible, then it will be used by default as the actual subprogram.

Generic instances of other generic packages

- A generic formal can be a package; it must be an instance of a generic package, so that the generic knows the interface exported by the package:

```
generic
with package P is new Q (<>);
```

- the actual must be an instance of the generic package Q
- the box after Q means that we do not care which actual generic parameters were used to create the actual for P

Generic package parameter

- It is possible to specify the exact parameters, or to specify that the defaults must be used
- The generic sees both the public part and the generic parameters of the actual package

```
generic
-- P1 must be an instance of Q with the
specified actual parameters:
with package P1 is new Q (Param1 => X,
Param2 => Y);
-- P2 must be an instance of Q where the
actuals are the defaults:
with package P2 is new Q;
```

Instantiating generics

- to instantiate a generic unit, use the keyword **new**:
- the generic formal types define *completely* which types are acceptable as actuals
- Ada requires that all instantiations be explicit.
- it is not possible to create special-case instances of a generic
- the object code can be shared by all instances of a generic
- when reading programs written by other people, there are no hidden instantiations and no special cases

Linear search

- Implement the linear search using generics
- Parameters: element, index, array type and a condition
- The generic should be procedure
- An out parameter should indicate if there is an element of the given condition, and which one is the first

Linear search

```
generic
    type Elem is private;
    type Index is (<>);
    type T is array ( Index range <> ) of Elem;
    with function Prop( A: Elem ) return Boolean;
procedure Linker (x: T; b: out Boolean; j: out Index);
```

Linear search

```
procedure Linker (x: T; b: out Boolean; j: out Index) is
begin
    b:= false;
    for i in reverse x'range loop
        if Prop(x(i)) then b:= true; j:= i; end if;
    end loop;
end linker;
```

Linear search - demo

```
with linker, Ada.Text_IO;
use Ada.Text_IO;
procedure mainlinker is
    type Index is new Integer;
    type Elem is new Integer;
    type T is array (Index range <>) of Elem;
    function myprop (x: Elem) return Boolean is
        begin return (x<0); end myprop;
    k: Index; b: Boolean;
    a: T(1..5):=(1,2,3,4,5);
    a1: T(1..5):=(1,-2,3,-4,5);
    a2: T(1..5):=(1,2,3,4,-5);
procedure Mylinker is new linker (Elem, Index, T, myprop);
```

Linear search demo

```
begin
    mylinker(a, b, k);
    if b then Put_Line( Elem'Image(a(k)) );
        else Put_Line(„ no negativ elements "); end if;
   mylinker(a1, b, k);
   if b then Put_Line( Elem'Image(a1(k)) );
      else Put_Line(" no negativ elements "); end if;
   mylinker(a2, b, k);
   if b then Put_Line( Elem'Image(a2(k)) );
   else Put_Line(" no negativ elements "); end if;
end mainlinker;
```

Conditional maximum search

- Implement the conditional maxumim search
- Parameters: element, index, array type and the searched condition
- The generic should be procedure
- In an out parameter indicate if there is an element of the given condition, and which one is that

Conditional maximum search

```
generic
   type Elem is private;
   type Index is (<>);
   type TA is array (Index range <>) of Elem;
   with function Cond ( A: Elem ) return Boolean;
   with function "<" ( A, B: Elem ) return Boolean is <>;
procedure Max_Search ( T: in TA; V: out Boolean;
                                           Max: out Elem );
```

Max

```
procedure Max_Search ( T: in TA; V: out Boolean;
                                              Max: out Elem ) is
   Mh: Index;
begin
   V := False;
  for I in T'Range loop
     if Cond(T(I)) then
          if V then if T(Mh) < T(I) then Mh := I; end if;
          else V := True; Mh := I; end if; end if; end loop;
   Max := T(Mh);
end Max_Search;
```

Max demo

```
with Max_Search, Ada.Integer_Text_IO, Ada.Float_Text_IO;
use Ada.Integer_Text_IO, Ada.Float_Text_IO;
procedure Max_Demo is
     type T is array (Integer range <>) of Float;
     function Int ( A: Float ) return Boolean is
     begin return A = Float(Integer(A)); end Int;
     procedure Max is new Max_Search(Float, Integer, T, Int);
    A: T(1..10) := (1.4,5.2,3.6,7.0,2.0,65.5,3.0,56.0,2.0,56.0);
    F: Float; V: Boolean;
begin
Max(A,V,F);
if V then Put( F ); end if; end Max_Demo;
```

Map generic

```
generic
    type A is private;
    type B is private;
    type Index is (<>);
    type TA_Array is array ( Index range <> ) of A;
    type TB_Array is array ( Index range <> ) of B;
    with function Op(x: A) return B;
function Map(ta: TA_Array) return TB_Array;
```

Map generic

```
function Map(ta: TA_Array) return TB_Array is
    tb:TB_Array(ta'Range);
begin
for i in ta'Range loop
    tb(i):=op(ta(i));
end loop;
return tb;
end Map;
```

Map demo

```
with map, Ada.Text_IO;
use Ada.Text_IO;
procedure Map_demo is
    type t1 is array (Integer range <>) of Integer;
    type t2 is array (Integer range <>) of Float;
    function square (x: Integer) return Float is
        begin return Float(x*x); end square;
function my_map is new map(Integer, Float, Integer, t1, t2, square);
a: t1(1..5):=(1, 2, 3, 4, 5); b: t2(a'range);

begin b:=my_map(a);
for i in b'Range loop Put_Line(Float'Image(b(i))); end loop;
end Map_demo;
```

Reversal of an array

```
generic
    type Elem is private;
    type Index is (<>);
    type T is array(Index range <>) of Elem;
procedure reversal (a: in out T);
```

Reverse

```
procedure reversal (a: in out T) is
   i: Index:= a'First;
   j: Index:= a'Last;
   tmp : Elem;
begin
    while i<j loop
             tmp:=a(i);
             a(i):=a(j);
             a(j):=tmp;
  i:=Index'Succ(i);
j:=Index'Pred(j);
end loop; end reversal;
```

demo

```
with reversal, Ada.Text_IO; use Ada.Text_IO;
procedure reversalmain is
    type T1 is array (Integer range <>) of Integer;
procedure myreversal is new reversal(Integer, Integer, T1);
a: T1(10..15):=(1,2,3,4,5,6);
a1: T1(10..16):=(1,2,3,4,5,6,7);
a2: T1:=(1,2); a3: T1(1..1); a4: T1(1..0);
begin
myreversal(a);
for i in a'range loop Put_Line(Integer'Image(a(i))); end loop;
end reversalmain;
```

Sort – generic in generic

- Instantiate a generic in another one
- E.g. swap and max_pos should be used in sorting
- Before usage needs instantiation (even if we don't know the types)

Swap generic

```
generic
    type T is private;
procedure Swap ( A, B: in out T );

procedure Swap ( A, B: in out T ) is
    Tmp: T := A;
begin
   A := B;
   B := Tmp;
end Swap;
```

Max_Pos generic function

```
generic
    type Elem is limited private;
    type Index is (<>);
    type TA is array (Index range <>) of Elem;
    with function "<" ( A, B: Elem ) return Boolean is <>;
function Max_Pos ( T: TA ) return Index;
```

Max_Pos generic function

```
function Max_Pos ( T: TA ) return Index is
 Mh: Index := T'First;
begin
  for I in T'Range loop
     if T(Mh) < T(I) then Mh := I;
     end if;
  end loop;
  return Mh;
end Max_ Pos;
```

Generic in generic

```
with Max_Pos, Swap;
procedure Sort ( T: in out TA ) is
  procedure Swap_Elem is new Swap(Elem);
  function Max_Pos_TA is new Max_Pos(Elem, Index, TA);
    Mh: Index;
   begin
      for I in reverse T'Range loop
      Mh := Max_Pos_TA( T(T'First..I) );
      Swap_Elem( T(I), T(Mh) );
      end loop;
end Sort;
```

Sort demo

```
with Ada.Text_IO, Sort; use Ada.Text_IO;
procedure SortDemo is
   type TA is array (Character range <>) of Float;
   procedure R_N is new Sort(Float, Character, TA);
   procedure R_Cs is new Sort(Float, Character, TA, ">");
   T: TA := (3.0,6.2,1.7,5.2,3.9);
begin
   R_Cs(T);
   for I in T'Range loop
       Put_Line( Float'Image( T(I) ) );
   end loop;
end SortDemo;
```

Has repetition

- Implement the `Has_Repetition` generic function with an indefinit vector array type (and its element and index type)
- The function gets a vector and return a boolean value which is true if there is an i such that `v(i) = v(i+1)`
- Test the generic for all possible cases

Has repetition

```
generic
  type Elem is private;
  type Index is (<>);
  type Vector is array ( Index range <> ) of Elem;
function has_repetition( T: Vector) return Boolean;
```

Has repetition

```
function has_repetition( T: Vector) return Boolean is
begin
    if T'length> 1 then
        for i in T'First..Index'Pred(T'Last) loop
            if T(i) = T(Index'Succ(i)) then return True;
            end if;
        end loop;
    end if;
   return False;
end has_repetition;
```

Demo

```
with has_repetition, Ada.Text_IO; use Ada.Text_IO;
procedure demo is
    type TInt is array (Integer range <>) of Integer;
    function my_rep is new has_repetition(Integer, Integer, TInt);
   v1: TInt := (1,1,2,4,5,650);
   v2: TInt := (1,2,3,4,5,6);
   v3: TInt(1..1); --:= (1);
   v4: TInt := (1,2, 3,3,3,56);
   v5: TInt := (1,2, 3,56,56);

begin
v3(1):= 3;
put_line(Boolean'Image(my_rep(v1))); put_line(Boolean'Image(my_rep(v2)));
put_line(Boolean'Image(my_rep(v3))); put_line(Boolean'Image(my_rep(v4)));
put_line(Boolean'Image(my_rep(v5)));
end demo;
```