COSC 3319.01 MWF

Bryan Benham

10/19/2022

Lab 2: Option A

RESULTS

```
Initial List. SIZE:  5
Car with    3 doors made by Chev
Car with    2 doors made by Ford
Car with    4 doors made by Ford
Car with    2 doors made by GMC
Car with    2 doors made by RAM

List after deletion. SIZE:  4
Car with    3 doors made by Chev
Car with    4 doors made by Ford
Car with    2 doors made by GMC
Car with    2 doors made by RAM

FINAL PRODUCT:
Plane with    4 doors,    4 engines, made by Cessna
Plane with    2 doors,    1 engines, made by Piper
Plane with    3 doors,    6 engines, made by Boeing
Car with    3 doors made by Chev
Car with    4 doors made by Ford
Car with    2 doors made by GMC
Car with    2 doors made by RAM
```

CODE

```
-- In file AbstStck.adb
package body AbstStck is
procedure insertFront(Stack: access AbstractStack; Y: in
AbstractStackElementPtr) is
    Pt: AbstractStackElementPtr;
  begin
      if Stack.Count = 0 then
         Y.Next := Stack.Top;
         Stack.Top := Y;
         Y.Prev := Stack.Bot;
         Stack.Bot := Y;
      else
         Y.Prev := Stack.Top;
         Stack.Top.Next := Y;
         Stack.Top := Y;
      end if;
      Stack.Count := Stack.Count + 1;
   end insertFront;

   procedure insertRear(Stack: access AbstractStack; Y: in
AbstractStackElementPtr) is
      Pt: AbstractStackElementPtr;
   begin
      if Stack.Count = 0 then
         Y.Next := Stack.Top;
         Stack.Top := Y;
```

```
         Y.Prev := Stack.Bot;
         Stack.Bot := Y;
      else
         Y.Next := Stack.Bot;
         Stack.Bot.Prev := Y;
         Stack.Bot := Y;
      end if;
      Stack.Count := Stack.Count + 1;
   end insertRear;

  function Pop(Stack: access AbstractStack) return AbstractStackElementPtr is
   Pt: AbstractStackElementPtr;
  begin
   if Stack.Top = null then -- Check for underflow.
     return null;
   end if;
   Stack.Count := Stack.Count - 1;
   Pt := Stack.Top;  Stack.Top := Stack.Top.Prev;  -- Pop stack, note
hemmoraging.
   return Pt;    -- Storage should be returned to an available storage list
for applications
   end Pop;     -- with high activity or executing for extended periods of
time.

  function StackSize(Stack: AbstractStack) return integer is
  begin  return Stack.Count;  end StackSize;
end AbstStck;
```

```
--  In file AbstStck.ads,  Creation of abstract stack.
package AbstStck is
  type AbstractStack is private;
  type AbstractStackElement is tagged private;
  type AbstractStackElementPtr is access all AbstractStackElement'Class;

   procedure insertFront(Stack: access AbstractStack; Y: in
AbstractStackElementPtr);
   procedure insertRear(Stack: access AbstractStack; Y: in
AbstractStackElementPtr);
   function Pop(Stack: access AbstractStack) return AbstractStackElementPtr;
   function StackSize(Stack: AbstractStack) return integer;

private
  type AbstractStackElement is tagged --Allow for heterogeneous stacks via
inheritance.
      record
         Next: AbstractStackElementPtr; --points to next (RLink)
         Prev: AbstractStackElementPtr; --points to previous (LLink)
    end record;

  type AbstractStack is
    record
      Count: integer := 0;  -- used to track the number of items in stack.
         Top: AbstractStackElementPtr := null; --top of stack
         Bot: AbstractStackElementPtr := null; --bottom of stack
    end record;
end AbstStck;
--------------------------------------------------------------------------
```

```
-- in file MakeCar.adb
with Ada.Text_IO; use Ada.Text_io;
with AbstStck;
package body MakeCar is
  package IntIO is new Ada.Text_IO.Integer_IO(Integer);  use IntIO;

  procedure AssignNumDoors(aCar: in out Car; N: in integer) is
  begin aCar.NumDoors := N; end AssignNumDoors;

  procedure AssignManufacturer(aCar: in out Car; Manu: in String4) is
   begin aCar.Manufacturer := Manu; end AssignManufacturer;

   function identifyFord(aCar: in Car) return boolean is
   begin
      if aCar.Manufacturer = "Ford" then
         return true;
         else return false;
      end if;
   end identifyFord;

  procedure PrintNumDoors(aCar: in Car) is
  begin put("Num doors = "); put(aCar.NumDoors); new_line; end PrintNumDoors;

  procedure PrintString4(PrtStr: String4) is
  begin for I in 1.. 4 loop
     put(PrtStr(I));
  end loop; end PrintString4;

  procedure PrintManufacturer(aCar: in Car) is
```

```
  begin put("Manufacturer is "); PrintString4(aCar.Manufacturer); new_line;
end;

  procedure IdentifyVehicle(aCar: in Car) is
  begin
    put("Car with "); put(aCar.NumDoors, 4); put(" doors");
    put(" made by "); PrintString4(aCar.Manufacturer); new_line;
  end IdentifyVehicle;
end MakeCar;
-------------------------------------------------------------------------
-- in file MakeCar.ads
with AbstStck;
package MakeCar is

  type String4 is new String(1..4);
  type Car is new AbstStck.AbstractStackElement with record
     NumDoors:  integer;
      Manufacturer: String4 := "GMC ";  -- Sample default value.

  end record;

  procedure AssignNumDoors(aCar: in out Car; N: in integer);
  procedure AssignManufacturer(aCar: in out Car; Manu: in String4);
  procedure PrintNumDoors(aCar: in Car);
  procedure PrintManufacturer(aCar: in Car);
  procedure IdentifyVehicle(aCar: in Car);
  function IdentifyFord(aCar: in Car) return boolean;
end MakeCar;
-------------------------------------------------------------------------
```

```
-- In file MakePlane.adb
with Ada.Text_IO; use Ada.Text_io;  with AbstStck;
package body MakePlane is
  package IntIO is new Ada.Text_IO.Integer_IO(Integer);  use IntIO;
  procedure AssignNumDoors(aPlane: in out Plane; N: in integer) is
  begin aPlane.NumDoors := N; end AssignNumDoors;
  procedure AssignManufacturer(aPlane: in out Plane; Manu: in String8) is
  begin aPlane.Manufacturer := Manu; end AssignManufacturer;

  procedure AssignNumEngines(aPlane: in out Plane; NE: in integer) is
  begin aPlane.NumEngines := NE; end AssignNumEngines;

  procedure PrintString8(PrtStr: String8) is
  begin for I in 1..8 loop  put(PrtStr(I));   end loop; end PrintString8;

  procedure PrintPlane(aPlane: in Plane) is
  begin
        put("Num doors for plane = "); put(aPlane.NumDoors, 4); new_line;
        put("Number engines = "); put(aPlane.NumEngines); new_line;
        put("Manufacturer = "); PrintString8(aPlane.Manufacturer); new_line;
  end PrintPlane;

  procedure IdentifyVehicle(aPlane: in Plane) is
  begin
    put("Plane with "); put(aPlane.NumDoors, 4); put(" doors, ");
    put(aPlane.NumEngines, 4); put(" engines, made by ");
    PrintString8(aPlane.Manufacturer); new_line;
  end IdentifyVehicle;
end MakePlane;
```

```
-------------------------------------------------------------------------
--  file MakePlane.ads:  Create planes for use with heterogeneous container.
with AbstStck;
package MakePlane is
  type String8 is new String(1..8);

  type Plane is new AbstStck.AbstractStackElement with record
     NumDoors:  integer;
     NumEngines:  integer;
     Manufacturer: String8 := "Boeing  ";
  end record;

  procedure AssignNumDoors(aPlane: in out Plane; N: in integer);
  procedure AssignManufacturer(aPlane: in out Plane; Manu: in String8);
  procedure AssignNumEngines(aPlane: in out Plane; NE: in integer);
  procedure PrintPlane(aPlane: in Plane);
  procedure IdentifyVehicle(aPlane: in Plane);
end MakePlane;
-----------------------------------------------------------------
```

```
with Ada.Text_IO; use Ada.Text_io;
with AbstStck; use AbstStck;
with MakeCar, MakePlane; use MakeCar, MakePlane;

procedure  UAbstSt2 is
  type Stack_Ptr is access AbstractStack;
   VehicleStack:  Stack_Ptr := new AbstractStack;
   stackCopy: Stack_Ptr := new AbstractStack;
   StackPoint: Stack_Ptr;
   success: integer;
   NewCar, CarPt, NewPlane, PlanePt, VehiclePt: AbstractStackElementPtr;

begin
   NewCar := new Car'(AbstractStackElement with 4, "Ford");  -- insert 4 door
Ford!
   insertRear(VehicleStack, NewCar); -- 1st car. Rear
   NewCar := new Car'(AbstractStackElement with 2, "Ford"); -- insert 2 door
Ford!
   insertFront(VehicleStack, NewCar); -- 2nd car. Front
   NewCar := new Car'(AbstractStackElement with 2, "GMC "); -- insert 2 door
GMC!
   insertRear(VehicleStack, NewCar); -- 3rd car. Rear
   NewCar := new Car'(AbstractStackElement with 2, "RAM "); -- insert 2 door
RAM!
   insertRear(VehicleStack, NewCar); -- 4th car. Rear
   NewCar := new Car'(AbstractStackElement with 3, "Chev"); -- insert 3 door
Chevy!
   insertFront(VehicleStack, NewCar); -- 5th car. Front
```

```
   put("Initial List. SIZE: ");
put(Integer'Image(StackSize(VehicleStack.all)));
   new_line; --check: should be 5

   for I in 1..StackSize(VehicleStack.all) loop
    VehiclePt := pop(VehicleStack);
    if VehiclePt.all in Car then  -- ** Identify class of object at run time.
         IdentifyVehicle(Car'Class(VehiclePt.all));
         if IdentifyFord(Car(VehiclePt.all)) = true and success /= 1 then
            success := 1;
         else
            insertRear(stackCopy, VehiclePt); --need to do rear so the 2nd
stack is in correct order
         end if;
    elsif VehiclePt.all in Plane then
         IdentifyVehicle(Plane'Class(VehiclePt.all));
         insertRear(stackCopy, VehiclePt);
    end if;
    new_line;
   end loop;

  put("List after deletion. SIZE: ");
put(Integer'Image(StackSize(stackCopy.all)));
   new_line; --check: size should be 4

   for I in 1..StackSize(stackCopy.all) loop
    VehiclePt := pop(stackCopy);
    if VehiclePt.all in Car then  -- ** Identify class of object at run time.
         IdentifyVehicle(Car'Class(VehiclePt.all));
```

```
    elsif VehiclePt.all in Plane then
      IdentifyVehicle(Plane'Class(VehiclePt.all));
      end if;
      insertRear(VehicleStack, VehiclePt);
    new_line;
   end loop;

  NewPlane := new Plane'(AbstractStackElement with 3, 6, "Boeing  ");  --
insert 3 door 6 engine Boeing!
  insertFront(VehicleStack, NewPlane);  --1st plane. Front
  NewPlane := new Plane'(AbstractStackElement with 2, 1, "Piper   "); --
insert 2 door 1 engine Piper!
  insertFront(VehicleStack, NewPlane); --2nd plane. Front
  NewPlane := new Plane'(AbstractStackElement with 4, 4, "Cessna  "); --insert
4 door 4 engine Cessna!
   insertFront(VehicleStack, NewPlane); --3rd plane. Front

   put("FINAL PRODUCT: "); new_line;

   for I in 1..StackSize(VehicleStack.all) loop
    VehiclePt := pop(VehicleStack);
    if VehiclePt.all in Car then  -- ** Identify class of object at run time.
         IdentifyVehicle(Car'Class(VehiclePt.all));
    elsif VehiclePt.all in Plane then
      IdentifyVehicle(Plane'Class(VehiclePt.all));
    end if;
    new_line;
   end loop;
end UAbstSt2;
```