Christopher McDaniel

COSC 3319.01

Submitted: 30 October 2019

Grading Option: "**A**"

"C" option transactions

Contents of Integer list from front to rear:

Integer: 95

Integer: 57

Integer: 33

Integer: 85

Integer: 62

Contents of Integer list from rear to front:

Integer: 62

Integer: 85

Integer: 33

Integer: 57

Integer: 95

Final contents of Integer list from front to rear:

Integer: 22

Integer: 85

"B" option transactions

Number of items in the below list: 5

Contents of Car list from front to rear:
Car with 3 doors is made by Chevy
Car with 2 doors is made by Ford
Car with 4 doors is made by Ford
Car with 2 doors is made by GMC
Car with 2 doors is made by RAM

Number of items in the below list: 4

Contents of Car list from front to rear:
Car with 3 doors is made by Chevy
Car with 4 doors is made by Ford
Car with 2 doors is made by GMC
Car with 2 doors is made by RAM

Number of items in the below list: 7

Final contents of Car and Plane list:
Plane with 4 doors and 4 engine(s) is made by Cessna
Plane with 2 doors and 1 engine(s) is made by Piper
Plane with 3 doors and 6 engine(s) is made by Boeing
Car with 3 doors is made by Chevy
Car with 4 doors is made by Ford
Car with 2 doors is made by GMC
Car with 2 doors is made by RAM

----------List_Package.ads----------

```
package List_Package is --Container stack, stack element, and element pointer types
   type Container is limited private;
   type CElement is tagged private;
   type CElementPtr is access all CElement'Class; --Allows access to CElement
                                                  --and any class inheriting
                                                  --from CElement.
   procedure InsertFront(list: access Container; amt: in CElementPtr);
   procedure InsertRear(list: access Container; amt: in CElementPtr);
   procedure Delete(list: access Container; amt: in out CElementPtr);
   procedure PrintItem(list: access Container; amt: CElementPtr);

   function FindItem(list: access Container; aValue: Integer) return CElementPtr;
   function NextItem(list: access Container; aValue: Integer) return CElementPtr;
   function ListSize(list: Container) return Integer;

private
   type CNodePtr is access CElement; --Define pointer type to Node.

   type Container is limited
      record
         Knt: Integer := 0; --Used to track the number of items in stack.
         Top, Bottom: CElementPtr := null; --Front and Rear
      end record;

   type CElement is tagged  --Allows for heterogenous stacks via inheritance
      record
         Value: Integer := 0;
         Next, Previous: CElementPtr;
      end record;
end List_Package;
```

----------MakeInteger.ads----------

```
with List_Package;

package MakeInteger is

   type MyInt is new List_Package.CElement with
      record
         Digit: integer;
      end record;

   procedure AssignDigit(aMyInt: in out MyInt; D: in integer);

   procedure IdentifyDigit(aMyInt: in MyInt);

end MakeInteger;
```

----------MakeCar.ads----------

```
with List_Package;

package MakeCar is

   type String5 is new String(1..5);

   type Car is new List_Package.CElement with
      record
         NumDoors: integer;
         Manufacturer: String5;
      end record;

   procedure AssignNumDoors(aCar: in out Car; N: in integer);

   procedure AssignManufacturer(aCar: in out Car; Manu: in String5);

   procedure IdentifyVehicle(aCar: in Car);

end MakeCar;
```

----------MakePlane.ads----------

```
with List_Package;

package MakePlane is
   type String8 is new String(1..8);

   type Plane is new List_Package.CElement with record
      NumDoors:  integer;
      NumEngines:  integer;
      Manufacturer: String8;
   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 IdentifyVehicle(aPlane: in Plane);

end MakePlane;
```

----------List_Package.adb----------

```
with Ada.Text_IO; use Ada.Text_IO;
with ada.Unchecked_Deallocation;
with Unchecked_Conversion;
with MakeCar, MakePlane, MakeInteger; use MakeCar, MakePlane, MakeInteger;

package body List_Package is

   function EmptyBuffer is new Unchecked_Conversion(CElementPtr, CNodePtr);

   --Provide opportunity for garbage collection and reuse of Nodes.
   procedure Free is new Ada.Unchecked_Deallocation(CElement, CNodePtr); --Reclaim heap
storage

   --Gets list size
   function ListSize (list: Container) return Integer is
   begin
      return List.Knt;
   end ListSize;

   --This will insert an element into the front of the list
   procedure InsertFront(list: access Container; amt: in CElementPtr) is
      PtrTop, PtrMain : CElementPtr := List.Top;
      IfKnt, ValueKnt : Integer := 0;

   begin
      if List.Top /= null then
         amt.Next := List.Top;
         List.Top.Previous := amt;
         List.Top := amt;
```

```
      amt.Previous := null;
   else
      List.Top := amt;
      List.Bottom := List.Top;
      amt.Previous := null;
      amt.Next := null;
   end if;

   List.Knt := List.Knt + 1;
   IfKnt := IfKnt + 1;
   amt.Value := 1;
   PtrTop := amt;
   PtrMain := amt;

   --Set index Value
   while ValueKnt < IfKnt
   loop
      PtrTop := PtrTop.Next;
      ValueKnt := ValueKnt + 1;
   end loop;

   while PtrTop /= null
   loop
      PtrMain := PtrMain.Next;
      PtrTop := PtrTop.Next;
      PtrMain.Value := PtrMain.Value + 1;
   end loop;
end InsertFront;

--This will insert an element into the rear of the list
procedure InsertRear(list: access Container; amt: in CElementPtr) is
```

```
begin
   if List.Top /= null then
      amt.Previous := List.Bottom;
      List.Bottom.Next := amt;
      List.Bottom := amt;
      amt.Next := null;
   else
      List.Bottom := amt;
      List.Top := List.Bottom;
      amt.Next := null;
      amt.Previous := null;
   end if;

   List.Knt := List.Knt + 1;

   --Set index Value
   amt.Value := List.Knt;
end InsertRear;

--This will delete an element from the Doubly Linked list
procedure Delete(list: access Container; amt: in out CElementPtr) is
   PtrTop: CElementPtr := List.Top;
begin
   if amt.Previous /= null then
      amt.Previous.Next := amt.Next;
   else
      List.Top := amt.Next;
   end if;

   if amt.Next /= null then
      amt.Next.Previous := amt.Previous;
```

```
   else
      List.Bottom := amt.Previous;
   end if;

   List.Knt := List.Knt - 1;

   PtrTop := amt;

   for I in amt.Value..List.Knt
   loop
      PtrTop := PtrTop.Next;
      PtrTop.Value := PtrTop.Value - 1;
   end loop;
end Delete;

--This will print the element the pointer is pointing to
procedure PrintItem (list: access Container; amt: CElementPtr) is
   PtrTop: CElementPtr := List.Top;
   Knt: Integer := 0;
begin
   if amt.all in MyInt then
      IdentifyDigit(MyInt'Class(amt.all));
   elsif amt.all in Car then
      IdentifyVehicle(Car'Class(amt.all));
   else
      IdentifyVehicle(Plane'Class(amt.all));
   end if;
   new_line(1);
end PrintItem;

--This will Find an element in the list
```

```
function FindItem(list: access Container; aValue: Integer) return CElementPtr is
   PtrTop, PtrMain : CElementPtr := List.Top;
   Knt: Integer := 0;
begin
   if List.Top = null then
      return null; --Checks for underflow
   else
      while Knt < aValue
      loop
         PtrMain := PtrMain.Next;
         Knt := Knt + 1;
      end loop;
      while PtrMain /= null
      loop
         PtrMain := PtrMain.Next;
         PtrTop := PtrTop.Next;
      end loop;
      return PtrTop;
   end if;
end FindItem;

--This will find and return the following pointer of an item
function NextItem(list: access Container; aValue: Integer) return CElementPtr is
   PtrTop, PtrMain: CElementPtr := List.Top;
   Knt: Integer := 0;
begin
   if List.Top = null then
      return null; --Checks for underflow
   else
      while aValue > Knt
      loop
```

```
            PtrMain := PtrMain.Next;
            Knt := Knt + 1;
         end loop;
         while PtrMain /= null
         loop
            PtrMain := PtrMain.Next;
            PtrTop := PtrTop.Next;
         end loop;
         return PtrTop.Next;
      end if;
   end NextItem;
end List_Package;
```

----------MakeInteger.adb----------

```
with Ada.Text_IO; use Ada.Text_io;
with List_Package;

package body MakeInteger is
   package IntIO is new Ada.Text_IO.Integer_IO(Integer);  use IntIO;

   procedure AssignDigit(aMyInt: in out MyInt; D: in integer) is
   begin
      aMyInt.Digit := D;
   end AssignDigit;

   procedure IdentifyDigit(aMyInt: in MyInt) is
   begin
      put("Integer: "); put(aMyInt.Digit, 4);
   end IdentifyDigit;

end MakeInteger;
```

----------MakeCar.adb----------

```
with Ada.Text_IO; use Ada.Text_io;
with List_Package;

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 String5) is
   begin
      aCar.Manufacturer := Manu;
   end AssignManufacturer;

   procedure PrintString5(PrtStr: String5) is
   begin for I in 1.. 5
      loop
         put(PrtStr(I));
      end loop;
   end PrintString5;

   procedure IdentifyVehicle(aCar: in Car) is
   begin
      put("Car with"); put(aCar.NumDoors, 4); put(" doors is");
      put(" made by "); PrintString5(aCar.Manufacturer);
      new_line(1);
   end IdentifyVehicle;
end MakeCar;
```

----------MakePlane.adb----------

```
with Ada.Text_IO; use Ada.Text_io;
with List_Package;

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 IdentifyVehicle(aPlane: in Plane) is
   begin
```

```
      put("Plane with"); put(aPlane.NumDoors, 4); put(" doors and");
      put(aPlane.NumEngines, 4); put(" engine(s) is made by ");
      PrintString8(aPlane.Manufacturer);
      new_line(1);
   end IdentifyVehicle;

end MakePlane;
```

----------List_Package.adb----------

```
with Ada.Text_IO, Ada.Integer_Text_IO; use Ada.Text_IO;
with List_Package; use List_Package;
with MakeCar, MakePlane, MakeInteger; use MakeCar, MakePlane, MakeInteger;

procedure Main is
   type CPointer is access Container;
   VehicleStack, IntegerStack: CPointer := new Container;
   NewMyInt, NewCar, NewPlane, IntegerPt, VehiclePt: CElementPtr;

   NewArray: Array(1..ListSize(VehicleStack.all)) of CPointer;
   NewArray1: Array(1..ListSize(IntegerStack.all)) of CPointer;

begin
   --C option data
   --a)   Insert 33 in front (right).
   NewMyInt := new MyInt'(CElement with 33);
   InsertFront(IntegerStack, NewMyInt);

   --b)   Insert 57 in front (right)
   NewMyInt := new MyInt'(CElement with 57);
   InsertFront(IntegerStack, NewMyInt);

   --c)   Insert 85 at the rear (left).
   NewMyInt := new MyInt'(CElement with 85);
   InsertRear(IntegerStack, NewMyInt);

   --d)   Insert 62 at the rear (left).
   NewMyInt := new MyInt'(CElement with 62);
   InsertRear(IntegerStack, NewMyInt);
```

```
--e)  Insert 95 at the front (right).
NewMyInt := new MyInt'(CElement with 95);
InsertFront(IntegerStack, NewMyInt);

--f)  Print the contents of the list from front (right) to rear (left).
put_line("Contents of Integer list from front to rear: ");
for I in reverse 1..ListSize(IntegerStack.all)
loop
  IntegerPt  := FindItem(IntegerStack, I);
  PrintItem(IntegerStack, IntegerPt);
end loop;
new_line(2);

--g)  Print the content of the list from rear (left) to front (right).
put_line("Contents of Integer list from rear to front: ");
for I in 1..ListSize(IntegerStack.all)
loop
  IntegerPt  := FindItem(IntegerStack, I);
  PrintItem(IntegerStack, IntegerPt);
end loop;
new_line(2);

--h)  Find and delete the node containing 57.
IntegerPt := FindItem(IntegerStack, 4);
Delete(IntegerStack, IntegerPt);

--i)  Find and delete the node containing 33.
IntegerPt := FindItem(IntegerStack, 3);
Delete(IntegerStack, IntegerPt);

--k)  Find and delete the node containing 62.
```

```
IntegerPt := FindItem(IntegerStack, 1);
Delete(IntegerStack, IntegerPt);

--l)    Insert 22 in front (right).
NewMyInt := new MyInt'(CElement with 22);
InsertFront(IntegerStack, NewMyInt);

--m)  Delete the node containing 95.
IntegerPt := FindItem(IntegerStack, 2);
Delete(IntegerStack, IntegerPt);

--n)   Print the contents of the list from front (right) to rear (left).
put_line("Final contents of Integer list from front to rear: ");
for I in reverse 1..ListSize(IntegerStack.all)
loop
   IntegerPt  := FindItem(IntegerStack, I);
   PrintItem(IntegerStack, IntegerPt);
end loop;
new_line(2);

--B option data
--Cars
--a) Insert a Ford with 4 doors at the rear (left).
NewCar := new Car'(CElement with 4, "Ford ");
InsertRear(VehicleStack, NewCar);

--b)   Insert a Ford with 2 doors at the front (right).
NewCar := new Car'(CElement with 2, "Ford ");
InsertFront(VehicleStack, NewCar);

--c) Insert a GMC with 2 doors at the rear (left).
```

```
NewCar := new Car'(CElement with 2, "GMC  ");
InsertRear(VehicleStack, NewCar);

--d) Insert a RAM with 2 doors at the rear (left).
NewCar := new Car'(CElement with 2, "RAM  ");
InsertRear(VehicleStack, NewCar);

--e) Insert a Chevy with 3 doors at the front (right).
NewCar := new Car'(CElement with 3, "Chevy");
InsertFront(VehicleStack, NewCar);

--f) Print the number of items in the list.
put_line("Number of items in the below list: " & Integer'Image(ListSize(VehicleStack.all)));
new_line(1);

--g) Print the contents of the list (front (right) to rear (left)).
put_line("Contents of Car list from front to rear: ");
for I in reverse 1..ListSize(VehicleStack.all)
loop
   VehiclePt  := FindItem(VehicleStack, I);
   PrintItem(VehicleStack, VehiclePt);
end loop;
new_line(2);

--h) Find and delete the first Ford in the list (search front (right) to rear (left)).
VehiclePt := FindItem(VehicleStack, 4);
Delete(VehicleStack, VehiclePt);

--i) Print the number of items in the list.
put_line("Number of items in the below list: " & Integer'Image(ListSize
        (VehicleStack.all)));
```

```
new_line(1);

--j) Print the contents of the list (front (right) to rear (left)).
put_line("Contents of Car list from front to rear: ");
for I in reverse 1..ListSize(VehicleStack.all)
loop
   VehiclePt := FindItem(VehicleStack, I);
   PrintItem(VehicleStack, VehiclePt);
end loop;
new_line(2);

--Planes
--k) Insert a plane with 3 doors and 6 engines by Boeing at the front (right).
NewPlane := new Plane'(CElement with 3, 6, "Boeing  ");
InsertFront(VehicleStack, NewPlane);

--l) Insert a plane with 2 doors and 1 engine by Piper at the front (right).
NewPlane := new Plane'(CElement with 2, 1, "Piper   ");
InsertFront(VehicleStack, NewPlane);

--m) Insert a plane with 4 doors and 4 engines by Cessna at the front (right).
NewPlane := new Plane'(CElement with 4, 4, "Cessna  ");
InsertFront(VehicleStack, NewPlane);

--Not an option, but performed for output conformity
put_line("Number of items in the below list: " & Integer'Image(ListSize
        (VehicleStack.all)));
new_line(1);

--n) Print the final list.
put_line("Final contents of Car and Plane list: ");
```

```
   for I in reverse 1..ListSize(VehicleStack.all)
   loop
      VehiclePt := FindItem(VehicleStack, I);
      PrintItem(VehicleStack, VehiclePt);
   end loop;

end Main;
```