Lab 3

Date Submitted: 03-29-2023

Name: Dylan Brown

COSC 3319 Section 01

Meeting Days: MWF

Grading Option: A

IO Redirection used:

linkedlisttransactions<test.txt>results

Results for my Lab

List size: 5

list Contents:

Car made by: Chev Number of Doors: 3

Car made by: Ford Number of Doors: 2

Car made by: Ford Number of Doors: 4

Car made by: GMC Number of Doors: 2

Car made by: RAM Number of Doors: 2

List size: 4

List Content:

Car made by: Chev Number of Doors: 3

Car made by: Ford Number of Doors: 4

Car made by: GMC Number of Doors: 2

Car made by: RAM Number of Doors: 2

List Contents:

Plane made by: Cessna Number of doors: 4 Number of Engines: 4

Plane made by: Piper Number of doors: 2 Number of Engines: 1

Plane made by: Boeing Number of doors: 3 Number of Engines: 6

Car made by: Chev Number of Doors: 3

Car made by: Ford Number of Doors: 4

Car made by: GMC Number of Doors: 2

Car made by: RAM Number of Doors: 2

LinkedList.ads

with Ada.Unchecked\_Deallocation;

generic

type element is tagged limited private;

with function "="(I,E: element'Class) return Boolean; --overloading = to compare equality for children

package Linkedlist is

type DList is private;

type elementPtr is access all element'Class;

procedure makeHead(list: in out DList);

function ListSize(list: in DList) return Integer;

procedure InsertAfter(list: in out DList; Item: in elementPtr);

procedure InsertBefore(list: in out DList; Item: in elementPtr);

generic

with procedure Process(I : element'Class);

procedure ManuealMove(list : in DList);

function FindItem(list : in DList; C : elementPtr) return elementPtr;

function Delete(list : in out DList; C : elementPtr) return elementPtr;

private

type Node;

type nodePtr is access Node;

type Node is record

Data : elementPtr;

Next, prev : nodePtr;

end record;

type DList is record

count : Integer := 0;

Head : nodePtr;

end record;

end Linkedlist;

LinkedList.adb

with Ada.Text\_IO; use Ada.Text\_IO;

with Ada.Unchecked\_Deallocation;

package body Linkedlist is

procedure Free is new Ada.Unchecked\_Deallocation(node, nodePtr);

procedure makeHead(list : in out DList) is -- makes an empty list that points back at intsself

Head : nodePtr := new node;

begin

list.count := 0;

Head.Next := Head;

list.Head := Head;

head.data := null;

head.prev := head;

end;

function ListSize(list: in DList) return Integer is

begin

return list.count;

end ListSize; -- get number of items in list

procedure InsertAfter(list: in out DList; Item: in elementPtr) is

newNode : nodePtr := new node;

begin

newNode.Data := Item;

newNode.next := list.head.next;

newNode.prev := list.head;

newNode.next.prev := newNode;

newNode.prev.next := newNode;

list.count := list.count + 1;

end;

procedure InsertBefore(list: in out DList; Item: in elementPtr) is

newNode : nodePtr := new node;

begin

newNode.Data := Item;

newNode.prev := list.head.prev;

newNode.next := list.head;

newNode.prev.next := newNode;

newNode.next.prev := newNode;

list.count := list.count + 1;

end;

procedure ManuealMove(list : in DList) is

newNode: nodePtr;

begin

NewNode := list.Head.Next;

while newNode /= list.Head loop

Process(newNode.Data.all);

newNode := newNode.Next; New\_Line;

end loop;

end;

function FindItem(list : in DList; C : elementPtr) return elementPtr is

newNode: nodePtr;

begin

newNode := list.Head.Next;

while newNode /= list.Head loop

if C.all = newNode.Data.all then

return newNode.Data;

end if;

newNode := newNode.Next;

end loop;

return null;

end;

function Delete(list : in out DList; C : elementPtr) return elementPtr is

newNode: nodePtr; pt : elementPtr;

begin

newNode := list.Head.Next;

while newNode /= list.Head loop

if C.all = newNode.Data.all then

newNode.prev.Next := newNode.Next;

newNode.Next.prev := newNode.prev;

pt := newNode.Data; Free(newNode);

list.count := list.count - 1;

return pt;

end if;

newNode := newNode.Next;

end loop;

return null;

end;

end Linkedlist;

MakeCar.ads

with Linkedlist;

package MakeCar is

type vehical is tagged null record; -- single class empty so have access to both child classes at once

function equality(V : in vehical; W : in vehical'Class) return Boolean;

procedure PrintItem(V : in vehical);

type Car is new vehical with record

NumDoors : Integer;

Manufacturer : String(1..4);

end record;

function UserString return string;

function NumberDoors return Integer;

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

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

procedure PrintDoors(aCar : in Car);

procedure PrintManufacturer(aCar : in Car);

procedure identify(aCar : in Car);

overriding procedure PrintItem(acar : in Car);

overriding function equality(aCar : in Car; V : in vehical'Class) return Boolean;

end MakeCar;

MakeCar.adb

with Ada.Text\_IO; use Ada.Text\_IO;

with linkedList;

package body MakeCar is

package intIo is new ada.Text\_IO.Integer\_IO(Integer);

use intIo;

function UserString return string is

t: string(1..4) := (others => ' ');

last : Integer;

begin

put("Enter Manufacturer: ");

Get\_Line(t,last);

return t;

end UserString;

function NumberDoors return Integer is

t : Integer;

begin

put("Enter number of Doors: ");

Get(t);

Skip\_Line;

return t;

end NumberDoors;

procedure PrintItem(V : in vehical) is --default print

begin Put\_Line("Much empty"); end PrintItem;

procedure AssignNumDoors(aCar : in out Car; N : in Integer) is

begin

aCar.NumDoors := N;

end AssignNumDoors;

procedure AssignManufacturer(aCar : in out Car; Manu: in String) is

begin

aCar.Manufacturer := manu;

end AssignManufacturer;

procedure PrintDoors(aCar : in Car) is

begin

put("Num of Doors");

put(aCar.NumDoors);

New\_Line;

end PrintDoors;

procedure PrintString4(PrtStr: String) 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 PrintManufacturer;

procedure identify(aCar : in Car)is

begin

put("Car with ");

put(aCar.NumDoors);

put(" doors");

put(" Made by ");

PrintString4(aCar.Manufacturer);

New\_Line;

end identify;

function equality(V : in vehical; W : in vehical'Class) return Boolean is --default is equal

begin

return False;

end equality;

overriding procedure PrintItem(aCar : in Car) is --print for planes

begin

Put("Car made by: "& acar.Manufacturer & " Number of Doors: " & aCar.NumDoors'Image);

end PrintItem;

overriding function equality(aCar : in Car; V : in vehical'Class) return Boolean is -- equal functions for cars

begin

return V in Car and then aCar.Manufacturer = Car(V).Manufacturer;

end equality;

end MakeCar;

MakePlane.ads

with linkedList;

with MakeCar; use MakeCar;

package MakePlane is

type Plane is new vehical with record

NumDoors : Integer;

NumEngines : Integer;

Manufacturer : String(1..8);

end record;

function UserString return string;

function NumberDoors return Integer;

function NumberEngines return Integer;

procedure AssignNumDoors(aPlane : in out Plane; N : in Integer);

procedure AssignManufacturer(aPlane : in out Plane; Manu : in String);

procedure AssignNumEngine(aPlane : in out Plane; NE : in Integer);

procedure PrintPlane(aPlane : in Plane);

procedure identify(aPlane: in Plane);

overriding procedure PrintItem(P : in Plane);

overriding function equality(P : in Plane; W : in vehical'Class) return Boolean;

end MakePlane;

MakePlane.adb

with linkedList;

with MakeCar; use MakeCar;

package MakePlane is

type Plane is new vehical with record

NumDoors : Integer;

NumEngines : Integer;

Manufacturer : String(1..8);

end record;

function UserString return string;

function NumberDoors return Integer;

function NumberEngines return Integer;

procedure AssignNumDoors(aPlane : in out Plane; N : in Integer);

procedure AssignManufacturer(aPlane : in out Plane; Manu : in String);

procedure AssignNumEngine(aPlane : in out Plane; NE : in Integer);

procedure PrintPlane(aPlane : in Plane);

procedure identify(aPlane: in Plane);

overriding procedure PrintItem(P : in Plane);

overriding function equality(P : in Plane; W : in vehical'Class) return Boolean;

end MakePlane;

linkedListTransactions.adb (Driver)

with Linkedlist;

with MakeCar, MakePlane; use MakeCar, MakePlane;

with Ada.Unchecked\_Deallocation;

with Ada.Text\_IO; use Ada.Text\_IO;

procedure linkedlisttransactions is

type DLPtr is access all vehical'Class;

procedure Free is new Ada.Unchecked\_Deallocation(vehical'Class, DLPtr);

function isEqual(V : in vehical'Class; W : in vehical'Class) return Boolean is

begin

return equality(V, W); -- returns the result of the overridden equality function it needs to use

end;

procedure PrintedItem(pItem : vehical'Class) is

begin

PrintItem(pItem); --prints the items depending on the overriden funciton it uses

end;

package VehicleList is new Linkedlist(vehical, isEqual);

use VehicleList;

procedure PrintList is new ManuealMove(PrintedItem);

list : DList;

doors:Integer;

engines:Integer;

manufactCar:string(1..4);

ManufactPlane:string(1..8);

V : access vehical'Class;

C : access Car;

P : access Plane;

K : aliased Car; --allows access to the values in car so it can actually search for the correct cars

begin

makeHead(list); -- begin with null head node

ManufactCar := MakeCar.UserString; --input manufact

doors := MakeCar.NumberDoors; --input num of doors

C := new Car'(vehical with doors , manufactCar );

InsertBefore(list, elementPtr(C));

ManufactCar := MakeCar.UserString; --input manufact

doors := MakeCar.NumberDoors; --input num of doors

C := new Car'(vehical with doors , manufactCar);

InsertAfter(list, elementPtr(C));

ManufactCar := MakeCar.UserString; --input manufact

doors := MakeCar.NumberDoors; --input num of doors

C := new Car'(vehical with doors , manufactCar);

InsertBefore(list, elementPtr(C));

ManufactCar := MakeCar.UserString; --input manufact

doors := MakeCar.NumberDoors; --input num of doors

C := new Car'(vehical with doors , manufactCar);

InsertBefore(list, elementPtr(C));

ManufactCar := MakeCar.UserString; --input manufact

doors := MakeCar.NumberDoors; --input num of doors

C := new Car'(vehical with doors , manufactCar);

InsertAfter(list, elementPtr(C));

Put\_Line("List size: " & ListSize(list)'Img);

Put\_Line("list Contents: ");

PrintList(list); New\_Line;

ManufactCar := MakeCar.UserString;

Skip\_Line;--input manufact

K := Car'(-1, ManufactCar);

V := FindItem(list, K'Access);

V := Delete(list, K'Access);

Put\_Line("List size: " & ListSize(list)'Img); New\_Line;

Put\_Line("List Content: ");

PrintList(list); New\_Line;

ManufactPlane := MakePlane.UserString; --input manufact

doors := MakePlane.NumberDoors; --input num of doors

engines := makePlane.NumberEngines;

P := new Plane'(vehical with doors, engines, ManufactPlane);

InsertAfter(list, elementPtr(P));

ManufactPlane := MakePlane.UserString; --input manufact

doors := MakePlane.NumberDoors; --input num of doors

engines := makePlane.NumberEngines;

P := new Plane'(vehical with doors, engines, ManufactPlane);

InsertAfter(list, elementPtr(P));

ManufactPlane := MakePlane.UserString; --input manufact

doors := MakePlane.NumberDoors; --input num of doors

engines := makePlane.NumberEngines;

P := new Plane'(vehical with doors, engines, ManufactPlane);

InsertAfter(list, elementPtr(P));

Put\_Line("List Contents: ");

PrintList(list);

end linkedlisttransactions;