Date Submitted: 01-30-2023

Name: Dylan Brown

COSC 3319 Section 01

Meeting Days: MWF

Grading Option: C

New DoubleStack.ads:

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

generic

type message is private;

package DoubleStack is

procedure retrieveMessage(msg: out message);

function StackEmpty return Boolean;

procedure vegpush(msg: in message);

procedure meatpush(msg: in message);

function vegPop return message;

function meatPop return message;

function spaceAvailable return Boolean;

--Add method (function or procedure) for inserting at front of queue here and in body.

end DoubleStack;

New DoubleStack.adb:

package body DoubleStack is

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer);

use IntIO;

capacity: Natural := 27; -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* storage space.

meatTop : Natural range 0 .. capacity -1 := 0;

vegTop: Natural range 0 .. capacity-1 := capacity-1;

subtype slotindex is natural range 0..(capacity - 1); -- Natural implies >= 0.

box: array(slotindex) of message; -- circular buffer

procedure meatpush(msg: in message) is -- meat stack push procedure

begin

if meatTop < (vegTop -1) then

meatTop := meatTop +1;

box(meatTop) := msg;

else

put("Error - stack full");

end if;

end meatpush;

procedure vegpush(msg: in message) is --veg stack push prodecure

begin

if vegTop > (meatTop +1) then

vegTop := vegTop -1;

box(vegTop) := msg;

else

Put("Error - stack full");

end if;

end vegpush;

function meatPop return message is --meat pop function returning pop message

begin

if meatTop /= 0 then

meatTop := meatTop -1;

return box(meatTop +1);

else

Put("nothing to be sold");

return box(meatTop -1);

end if;

end meatPop;

function vegPop return message is -- veg pop returns message that poped

begin

if vegTop < capacity then

vegTop := vegTop +1;

return box(vegTop-1);

else

Put("Nothing to sell");

return box(vegTop-1);

end if;

end vegPop;

procedure retrieveMessage(msg: out message) is -- if there is meat to sell then it sells meat if not veg

begin

if meatTop > 0 then -- obtain meat product

msg := meatPop;

elsif vegTop < 21 then

msg := vegPop;

else

Put("Nothing to sell"); New\_Line;

end if;

end retrieveMessage;

function spaceAvailable return Boolean is -- if there is space then then it returns true

begin

if meatTop + (capacity - vegTop)>= capacity -1 then

return False;

else

return True;

end if;

end spaceAvailable;

function StackEmpty return Boolean is -- if it is empty then it returns true

begin

if meatTop /= 0 or vegTop /= capacity then

return False;

else

return True;

end if;

end StackEmpty;

end DoubleStack;

GateKeeper.ads Not Changed

New GateKeeper.adb:

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

with Ada.Calendar; use Ada.Calendar;

package body GateKeeperService is

package IntegerIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntegerIO;

task body GateKeeper is

package DoubleStacks is new DoubleStack (Food\_Pack); -- \*\* specify size for storage space. \*\*

use DoubleStacks;

meat : Integer :=0;

veg : Integer :=0;

numbersoldmeat : Integer :=0;

numbersoldveg : Integer := 0;

rejected: Integer := 0;

-- Declare food packet counters here.

Start\_Time: Ada.Calendar.Time;

End\_Time: Ada.Calendar.Time;

begin

delay 0.5; -- allow 1/2 hour to initialize facility.

Start\_Time := Ada.Calendar.Clock;

End\_Time := Start\_Time + 1.0 \* 8.0 \* 5.0; -- 1.0 sec./hour \* 8 hours/days \* 5 days

-- Termiate after lossing 5 customers or time to close has arrived.

while rejected < 5 and Ada.Calendar.Clock < End\_Time loop -- Termiate after lossing 5 customers

-- In Ada, a "select" statement with multiple "or" options must uniformly

-- process (randomly) the "accept" statements. This prevents any single

-- "accept" from starving the others from service.

--

-- Rules for "Select":

-- 1) If no task are waiting for service, the task sleeps.

-- 2) If only one of the "accept" entries has a task waiting that task is served.

-- 3) If sleeping and a task or tasks arrive simultaneously, awake a service the

-- the first arrival.

-- 4) If multiple "accepts" have task waiting, service them in random order

-- to prevent starvation.

--

select

-- new arrivals of food

accept acceptMessage( newFood: in Food\_Pack) do

if ( DoubleStacks.spaceAvailable ) then

if(getFood\_PackFoodType(newfood) in GrainVegetable)then --counts how much meat and veg are generated

DoubleStacks.vegpush(newfood);

PrintFood\_Pack( newFood ); new\_line;

veg := veg +1;

-- Ada.Text\_IO.Put\_Line(Integer'Image(veg));

else

DoubleStacks.meatpush(newfood);

PrintFood\_Pack( newFood ); new\_line;

meat := meat +1;

end if;

-- get food packet food type in food data structures

put("GateKeeper insert accepted ");

PrintFood\_Pack( newFood ); new\_line;

else

rejected := rejected + 1;

put(" Rejected by GateKeeper: "); new\_line;

PrintFood\_Pack( newFood ); new\_line;

put(" Rejected = "); put(rejected);

put(". Sent to another distribution facility!"); new\_line(3);

end if;

end acceptMessage;

or

-- Accept request for distribution from sales

accept retrieveMessage( newFood: out Food\_Pack; availableForShipment: out Boolean) do

availableForShipment := False;

if not(DoubleStacks.StackEmpty) then

availableForShipment := True;

DoubleStacks.retrieveMessage( newFood );

PrintFood\_Pack( newFood ); put(" Removed by GateKeeper for shipment."); new\_line;

if (getFood\_PackFoodType(newfood) in GrainVegetable)then -- this if counts how many meat and veg sold

numbersoldveg := numbersoldveg +1;

else numbersoldmeat := numbersoldmeat +1;

end if;

end if;

end retrieveMessage;

end select;

delay 1.1; -- Complete overhead due to accepting or rejecting a request prior to new iteration.

end loop;

put("total meat packets sold");

Ada.Text\_IO.Put\_Line(Integer'Image(numbersoldmeat)); --gives me total meat sold

put("total veg packets sold");

Ada.Text\_IO.Put\_Line(Integer'Image(numbersoldveg)); --gives me total veg sold

put("number of veg packets generated");

Ada.Text\_IO.Put\_Line(Integer'Image(veg)); --gives total veg generated

new\_Line(1); put("number of meat packets generated");

Ada.Text\_IO.Put\_Line(Integer'Image(meat)); --gives total meat generated

-- print time in service, statistics such as number food pacekets of meat and non-meat products processed.

new\_line(2); put("Hours of operation prior to closing: ");

Ada.Text\_IO.Put\_Line(Duration'Image(Ada.Calendar.Clock - Start\_Time)); new\_line(2);

end GateKeeper;

end GateKeeperService;

I/O redirection screenshot:



First 3 Hours:

How many Product Generators?

How many points of sale?

B delivered.

SQUASH B

GateKeeper insert accepted SQUASH B

Next grain shipment arrives 9.93067E-01 Time units!

B delivered.

B delivered.

RICE B

GateKeeper insert accepted RICE B

Next grain shipment arrives 1.27761E+00 Time units!

B delivered.

RICE B

GateKeeper insert accepted RICE B

Next grain shipment arrives 4.34619E-01 Time units!

B delivered.

B delivered.

TOMATO B

GateKeeper insert accepted TOMATO B

Next grain shipment arrives 8.03268E-01 Time units!

Last 2 Hours:

WHEAT B

GateKeeper insert accepted WHEAT B

Next grain shipment arrives 6.13403E-01 Time units!

M delivered.

RICE B

GateKeeper insert accepted RICE B

Next grain shipment arrives 2.33375E-02 Time units!

M delivered.

Rejected by GateKeeper:

CORN B

Rejected = 1. Sent to another distribution facility!

Next grain shipment arrives 1.31579E+00 Time units!

Rejected by GateKeeper:

FOWEL M

Rejected = 2. Sent to another distribution facility!

Next grain shipment arrives 1.22609E+00 Time units!

Hours of operation prior to closing: 41.035857900

Other Statistics:

Number of generators used: 3

Numbers of POS used: 2

Meat packets sold: 4

Other packets sold: 1

Total Packets Sold: 5