**Data Structures Lab 1 DS Fall 2022**

Lab1 is designed to meet the suggestions of industry speakers from last summer and this summer (2022) with respect to what they thought would help prepare students for jobs. In general, you will be asked to modify existing systems. You will typically not have complete knowledge of the entire system.

You should install an Ada compiler and development environment now. Use these tools to implement the program structure in Appendix A (page 16) in a dedicated directory. For example, you might name the directory “Lab1” or “FoodDistributionSystem.” I recommend you create the files in the order they appear in Appendix A. The basic rule is “something may not be referenced until what it references exists” when compiling or linking/building.

This system was developed using OOD (Object Oriented Design) and OOPS (Object Oriented Programming) hence most pieces of the system consist of a specification ( .ads ) followed by a body ( .adb ). Developers should only have to access the specification to know what a module does and the behavior they can directly access. They should not have to know or care how it is physically implemented in the body. Specifications must be compiled prior to compiling and building (linking) their bodies. You only compile specification files ( .ads ). You must compile/build (link) Ada bodies ( .adb ).

Compiling and/link individual files as appropriate. You may execute the system after building/linking “ProductDistributionMain.” For a first attempt, create 1 product generator and 1 point of sale when prompted at the terminal. Feel free to experiment with other values of these parameters.

IF YOU ALREADY HAVE A KNOWLEDGE OF ADA FEEL FREE TO WORK ON THE LAB. IF YOU DO NOT KNOW ADA THEN WAIT TILL IT IS ASSIGNED TO MAKE ANY CHANGES! We have completed everything in class to allow completion of the “A” Option.

Dr. B

COSC 3319 Data Structures

Lab 1 Fall 2022

**From:** Ima Yourboss

**To:** Loyal Employee

**Due:** The “D” option lab ***must*** be submitted prior **to the start of class October 19.** If you successfully submitted the “D” option lab you may submit the “C” option of the lab **prior to the start of class October 26 (Q drop deadline)**. If you successfully met the submission deadlines for the “D” and “C” option you may submitted the “**B and/or “A” options prior to the start of class November 2.**

You may submit a “higher” grading option at any time prior to the due date for the current grading option as long as you have met all deadlines for previously submitted grading options. **You are encouraged to submit the “A” option as soon as possible**. You may submit labs up to 5 times to improve your grade as long as you do so prior to the due date. For feedback, I must receive your lab sufficiently early to grade it on my schedule. Remember I have almost 150 students registered in classes this semester competing for my time.

**Expect additional labs to be assigned during this time frame. You may use Windows, UNIX (any flavor including Linux or Apple) or the Chrome operating system as long as they have an Ada compiler. Other operating systems will be considered on request.**

*This lab is meant to push you beyond your current level of expertise to new levels of accomplishment, i.e., professional grade. A goal is to force you to research and master new technology either on your own or with a minimum of external help. I am always available for questions. Labs must be submitted in an envelope sufficiently large to hold your submission without having to fold the lab materials. Write your name, “COSC 3319” and section number (time/day class meets) in the upper left hand corner of the envelope opposite the side away from the flap*. Manila envelopes (approximately 8-1/2 by 11 inches) work well. You may use the same envelope for all labs during the semester.

**You must include a cover page on the lab clearly stating THE DATE SUBMITTED, your name, COSC 3319, section number, day the class meets (MWF or TTH) and grading option you have completed.** Programs failing to state the grading option will be graded as "C" option programs. The cover page should be followed by the lab results, any data files you created, then required code. Do not embarrass yourself or me with screen shots unless screen shots are specifically specified. This lab is not technically challenging with respect to data structures. Its primary purpose is to ensure you gain the required background to complete “A” option work on future labs. Step up to “professional grade!”

Additional labs will be assigned prior to the due dates. Do not procrastinate finishing this first assignment. It will ensure you master the language technology required to implement the advanced data structures labs. This lab is designed so you can start by implementing the “D” option. The “D” option can be converted to the “C” option, then the “B” option, and finally the “A” option with minimal loss of previous effort. In essence, you tell me what you are worth. Be “Professional Grade!” Do the “A” option.

The first lab requires Ada. Remaining labs will give you a choice of languages. You are encouraged to use Ada for several reasons. First, if you ever work with a company contracting with DOD (Department of Defense – possibly the largest contractor in the USA), aircraft companies or European countries, knowledge of Ada is potentially an advantage. You should be able to use Ada, C++, C#, Java, or Groovy for most labs. Use of “C,” Java or Groovy will normally result in a “maximum 10 point implementation fee (IMF). Java and Groovy in particular lack the required ability to allocate memory etcetera in a real time environment with “hard real time constraints.” **Due to bad experiences from previous semesters, you may not use Python**. This class is oriented towards a very large group real time environments with some latitude. Python does not support the abstractions required for the advanced requirements on most labs. In addition, Python fails to adequately support large industrial group projects. Substantial examples using Ada will be utilized in class over the next few weeks. You may use Ada code developed in class to implement labs. **You may not use “maps,” “lists” or other prepackaged facilities in Ada or any other language on any lab during the semester.** Rather you will be learning to implement similar types of facilities tailored to requirements specific to your work environment after graduation.

If you work for bleeding edge technology companies, you will learn to change programming languages like you change your clothes. **There is no “best” language. Learn to select the most appropriate language for the task to be accomplished**. There are two required courses in Java (COSC1436 and COSC1437) and it is used in several other courses. “C/C++” is taught in COSC 2347 and used in several other courses. Ada will be used in this class, Software Engineering and Operating Systems when taught by me. The more languages you know prior to graduation, the easier it will be to learn new languages on the job after graduation. Most companies will expect you to learn new languages on your time using your own resources in two or three weeks (potentially less).

**Basic Information**

**Specification:**

We must develop software to manage an inter-planetary food distribution system for use in multiple regional solar systems. The basic software should consist of a food storage system (Warehouse) Gate Keeper (managed as a circular queue), and food sales/points-of-sale. Arriving food shipments will request storage space in the ware house (M 🡪1 relationship). If insufficient space is available, the food shipment should be routed to another system. Food Sales (N 🡪1) request access to food packets from the ware house (Gate Keeper), prepares the food for sale and completes the sales transaction. If the warehouse receives multiple simultaneous request for service (request for storage space) and/or multiple request for food packets (receive a packet from sales), the request must be serviced randomly (uniformly distributed fashion) to prevent starvation.

The Gate Keeper and Food Sales software will be used as the final system. The Product Generator software will be replaced in the final system by arriving food shipments. To test the design and determine system flexibility implementation must allow for multiple Product Generators (separate concurrent task/threads) and multiple Retail Sales (separate concurrent task/threads) communicating with a single Gate Keeper task/thread to prevent race conditions.

The basic system as seen by the specification team follows:

Communicating, Multi-Tasked Design

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Food Arrival Service** |  | **Warehouse Gate Keeper** |  | **Food Sales** |
|  | Determine arrival of food packets and request storage from Gate Keeper. | M-1 | Provide temporary storage for food shipments. Allow sales access to food. | 1-M | Request and obtain food from Gate Keeper for sale. |

**System Design Using OOD**

After consulting with those processing arriving food packets and sales, the following system structure has been specified allowing up to five groups to work on the system in parallel once the global classes/package specifications for Food\_DataStructures and Stats\_FoodDistribution are available.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | Food\_DataStructures.ads |  |  |  |  |  |  |  | Stats\_FoodDistribution.ads |  |  |
|  |  | Global food data |  |  |  |  |  |  |  | Global statistical distributions |  |  |
|  |  | Access methods |  |  |  |  |  |  |  | Access methods |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | Food\_DataStructures.adb |  |  |  |  |  |  |  | Stats\_FoodDistribution.adb |  |  |
|  |  | Implementation of data types and access methods |  |  |  |  |  |  |  | Implementation of statistical methods and access functions |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | “with” |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | **Distribution\_Service.ads** |  |  |  | **GateKeeper.ads** |  |  |  | **Food\_SalesService.ads** |  |  |
|  |  | task type Product\_Generator |  |  |  | task GateKeeper |  |  |  | task type RetailSales |  |  |
|  |  | Generate arrival of food packets statistically based on experience and ask for storage space. |  |  |  | entry accept(Message()  entry retrieveMessage() |  |  |  | Retrieve food packets from storage, package and sell to consumers. |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | Distribution\_Service.adb |  |  |  | GateKeeper.adb |  |  |  | SalesService.adb |  |  |
|  |  | Behavior implementation |  |  |  | Implementation of storage space. |  |  |  | Behavior implementation |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Temporary stub code to facilitate developing the Product Distribution system  Product\_DistributionMain.adb |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

The detailed **design specifications** for all classes/packages follow. Implementation of their bodies independent of the rest of the system should be possible.

The Gate Keeper is required to provide storage as a circular queue maximizing space utilization.

--in file CircularQue

--

--The GateKeeper requires a circular queue for storage and retrieval. The package has been

--made a generic to allow use in future systems with respect to the data type and queue size.

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

--\*\* This implementation of the circular queue sacrifices run-time allowing complete utillization of all queue space.

-- Message should not be sent without verifying the queue has avaliable storage space!!!!

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

generic

type message is private;

capacity: in Natural := 24; -- Default queue size

package CircularQue is

procedure acceptMessage(msg: in message);

procedure retrieveMessage(msg: out message);

function circularQueEmpty return Boolean;

function circularQueFull return Boolean;

end CircularQue;

Food packet contents and behavior for all modules.

Allow the user to create dynamically as many independent centers (tasks/threads) to accept arriving food products as required!

-- In file Food\_DataStructures.ads

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

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

with Ada.Numerics.Discrete\_Random;

package Food\_DataStructures is

--

-- Use 1.0 seconds of real time to simulate passage of 1 hour simulated time.

--

type Food\_Type is (Wheat, Corn, Rice, Potatoes, Squash, Tomato, Steak, Pork, Fish, Fowel);

subtype GrainVegetable is Food\_Type range Wheat .. Tomato;

package Food\_TypeIO is new Ada.Text\_IO.Enumeration\_IO(Food\_Type);

use Food\_TypeIO;

type Food\_Pack is private;

procedure PrintFood\_Pack( foodIn: in Food\_Pack );

procedure PrintFood\_PackType( FoodIn: in Food\_Pack );

procedure PrintFood\_PackShipment(FoodIn: in Food\_Pack);

function getFood\_PackFoodType( FoodIn: in Food\_Pack ) return Food\_Type;

function getFood\_PackFoodShipment( FoodIn: in Food\_Pack ) return Character;

procedure setFood\_PackFoodType( FoodIn: in out Food\_Pack; FoodType: Food\_Type );

procedure setFood\_PackShipment( FoodIn: in out Food\_Pack; FoodShipment: Character );

function RandomFoodType return Food\_Type;

private

type Food\_Pack is record

aFoodType: Food\_Type := Fowel;

aFoodShipment: Character := 'M';

end record;

end Food\_DataStructures;

Statistical distributions for the Gatekeeper and Sales.

-- in file Stats\_FoodDistribution.ads

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

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

with Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

use Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

with Food\_DataStructures; use Food\_DataStructures;

package Stats\_FoodDistribution is

--

-- Use 1.0 seconds of real time to simulate passage of 1 hour simulated time.

--

my\_Generator: Generator;

-- Exponential distribution using interpolation and famous data points from early IBM

-- Fortran statistical packages.

function Next\_Exponential return float;

-- Time requried to arrange raw food packets for sale.

function PrepareGrainVegetableFood\_PackforSales return duration;

function PrepareMeatFood\_PackforSales return duration;

end Stats\_FoodDistribution;

Allow the user to create dynamically as many independent concurrent tasks/threads as required to simulate the arrival of food freighters in the system.

-- in file Food\_SalesService.ads

--

-- The software suite consists of Product Distribution Main,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- The specification require the ability to create multiple points of sale.

with Food\_DataStructures; --use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

package Food\_SalesService is

task type RetailSales;

--task RetailSales; -- May used if only a simgle point of sales is require.

end Food\_SalesService;

The system will utilize a single gatekeeper to prevent race conditions and starvation with respect to sales and arriving food packets. \*\*

-- in file GateKeeperService.ads

--

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite

-- represents the software to manage an "embedded" planetary system

-- food receiving and distribution system. The Distribution\_Service

-- module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

-- The GateKeeper accepts Food\_Packs frfom inter-galatic transports

-- and repacks them into Food\_Packs suitable for Food\_SalesService to

-- distribute to the planets in the system. Food\_Packs are removed

-- from the default FIFO queue associated with the "accept" statement

-- and stored in a circular queue for distribution.

with Food\_DataStructures; use Food\_DataStructures;

With Stats\_FoodDistribution; use Stats\_FoodDistribution;

with CircularQue;

package GateKeeperService is

task GateKeeper is

-- accept Food\_Packs from interplanetary vesssels.

entry acceptMessage( newFood: in Food\_Pack );

--Allow sals to retrive the repackaged Food\_Packs.

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

end GateKeeper;

end GateKeeperService;

Allow the user to create dynamically as many independent concurrent arriving product distribution centers and/or sales (tasks/threads) as required!

-- in file Food\_SalesService.ads

--

-- The software suite consists of Product Distribution Main,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- The specification require the ability to create multiple points of sale.

--

with Food\_DataStructures; --use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

package Food\_SalesService is

task type RetailSales;

--task RetailSales; -- Definition may be used if only a single point of sales is required.

end Food\_SalesService;

####################################################################

**All Grading Options!**

Management expects all results to appear typed immediately after the Cover Page. *Management does not desire to see the entire transcript of events generated by the simulation*. Results should be summarized in an appropriate table for multiple runs with an appropriate evaluation of the results. You must establish your answer is the best using actual data (data driven decision making). **The simulation terminates when 5 food packet vessels are rejected by the Gate Keeper or the simulation exceeds 40 hours**. The Gate Keeper only checks for elapsed time after processing a request from a Product Generator to store a food packet or release a food packet to Sales. As a result the clock my drift higher than 40 hours.

**“D-”: Option (maximum grade is 55):**

Create the software system using Appendix A in the specified file locations then compile and link (build) the executables. The system was created using one second of simulation time to represent an hour of real time. Determine the minimum queue size to maximize the amount of food product sold during a 40 hour work week. This problem requires your cognitive insight. You may vary the size of the queue, number product generators and number of sales task. **You must execute the system from the command prompt to provide the number of product generators and sales tasks using I/O redirection for all input and output**. Your report must include the number of food generators used, number of sales task used, the number of meat food packets sold, other food packets sold and total food packets sold. Submit the system log for at least the first 2 hours of execution and 2 hours prior to the final results by copying the material from the log file created by your I/O redirection. Paste the results into your report. You need not include the entire result file.

**Sample Execution:**

**Contents of Data1.txt creating 1 product generator and 1 sales task:** I used notepad to create the data file but any ascii text editor (e.g., Textpad and Ada Guide) will work. You may not use word processors to create data files as they insert invisible formatting information in the file in addition to the values you desire. Remember the default for most languages is stream I/O. Separate values by white space (one or more spaces, tabs or newlines). The following example creates 1 food generation (receiving station for arriving vessels) and 1 sales center.

C:\Software\DataStructures\Lab2StarWarsB>notepad Data1.txt // Enter 1 <white space> 1.

C:\Software\DataStructures\Lab2StarWarsB>type Data1.txt // print the contents of Data1.txt on crt.

1 1

Using I/O redirection to read from file Data1.txt and print results in the ascii text file “Results.”

C:\Software\DataStructures\Lab2StarWarsB>ProductDistributionMain < Data1.txt > Results

To print the contents of “Results” on the screen use “type” for Microsoft or “cat” for Linux. You may copy from the file, text editor or screen to a word processor for your report.

C:\Software\DataStructures\Lab2StarWarsB type Results

How many Product Generators?

How many points of sale?

M delivered.

GateKeeper insert accepted PORK M

Next grain shipment arrives 1.60347E-01 Time units!

B delivered.

GateKeeper insert accepted CORN B

Next grain shipment arrives 2.90697E+00 Time units!

PORK M Removed by GateKeeper for shipment.

Retail Sales successfuly sold PORK M

CORN B Removed by GateKeeper for shipment.

B delivered.

Retail Sales successfuly sold CORN B

GateKeeper insert accepted WHEAT B

Next grain shipment arrives 8.04831E-02 Time units!

M delivered.

GateKeeper insert accepted FISH M

Next grain shipment arrives 6.27168E-02 Time units!

M delivered.

GateKeeper insert accepted STEAK M

Next grain shipment arrives 3.92095E-01 Time units!

B delivered.

GateKeeper insert accepted RICE B

Next grain shipment arrives 2.04101E+00 Time units!

WHEAT B Removed by GateKeeper for shipment.

B delivered.

GateKeeper insert accepted CORN B

Next grain shipment arrives 3.44041E-01 Time units!

B delivered.

GateKeeper insert accepted WHEAT B

Next grain shipment arrives 2.53420E+00 Time units!

Retail Sales successfuly sold WHEAT B

FISH M Removed by GateKeeper for shipment.

M delivered.

GateKeeper insert accepted FOWEL M

Next grain shipment arrives 1.00796E+00 Time units!

Retail Sales successfuly sold FISH M

STEAK M Removed by GateKeeper for shipment.

B delivered.

GateKeeper insert accepted WHEAT B

Next grain shipment arrives 1.77822E+00 Time units!

B delivered.

GateKeeper insert accepted RICE B

Next grain shipment arrives 4.77431E-01 Time units!

B delivered.

GateKeeper insert accepted RICE B

Next grain shipment arrives 2.88129E+00 Time units!

Retail Sales successfuly sold STEAK M

RICE B Removed by GateKeeper for shipment.

M delivered.

GateKeeper insert accepted FISH M

Next grain shipment arrives 3.08041E-01 Time units!

M delivered.

GateKeeper insert accepted PORK M

Next grain shipment arrives 6.52692E-01 Time units!

M delivered.

GateKeeper insert accepted FISH M

Next grain shipment arrives 4.37848E+00 Time units!

Retail Sales successfuly sold RICE B

CORN B Removed by GateKeeper for shipment.

Retail Sales successfuly sold CORN B

WHEAT B Removed by GateKeeper for shipment.

Retail Sales successfuly sold WHEAT B

FOWEL M Removed by GateKeeper for shipment.

B delivered.

Retail Sales successfuly sold FOWEL M

GateKeeper insert accepted WHEAT B

Next grain shipment arrives 1.15202E+00 Time units!

WHEAT B Removed by GateKeeper for shipment.

B delivered.

GateKeeper insert accepted RICE B

Next grain shipment arrives 5.86165E+00 Time units!

Retail Sales successfuly sold WHEAT B

RICE B Removed by GateKeeper for shipment.

Retail Sales successfuly sold RICE B

RICE B Removed by GateKeeper for shipment.

B delivered.

GateKeeper insert accepted RICE B

Next grain shipment arrives 3.29551E-01 Time units!

Retail Sales successfuly sold RICE B

M delivered.

Hours of operation prior to closing: 41.155000000 // Shows drift, most likely insufficient sales staff.

**“C”: Option (maximum grade is 75): \*\*\* You are only eligible to submit the “C” option if you met the deadline for the “D-“ option. \*\*\***

We currently have a shortage of meat products. Modify the generic circular queue “CircularQue.ads/adb” to allow insertion of arriving meat food packets at the front of the queue by the Gate Keeper so they will be sold prior to other food packets. Hint: Modify the generic “ads” and “adb” files by adding the new insertion method to be used by the Gate Keeper. You will find algorithms for these types of operation in the Data Structures notes for the case where we minimized time by sacrificing one unit of space in the queue. **You will have to modify this algorithm to match the algorithm in the existing code where we sacrificed time to maximize space utilization, i.e., we used all space in the queue**.

Create the software entire system with your modifications using Appendix A in the specified file locations then compile and link (build) the executables. The system was created using one second of simulation time to represent an hour of real time.

**Submission: First provide a copy or your modified code for the “CircularQue.ads/adb.” Determine the minimum queue size to maximize the amount of food product sold during a 40 hour work week. This problem requires your cognitive insight. You may vary the size of the queue, number product generators and number of sales task.** **You must execute the system from the command prompt using I/O redirection to provide the number of product generators and points-of-sale (sales staff/task)**. Your report must include the number of food generators used, number of sales task used, the number of meat food packets sold, other food packets sold and total food packets sold. Submit the results of at least the first 2 hours of execution and 2 hours prior to the final results by copying the material from the file created by your I/O redirection. Paste the results into your report. You need not include the entire result file.

**“B+: Option (maximum grade is 89): \*\*\*You are only eligible to submit the “B+ or higher option” only if you met the deadline for the “C“ option. \*\*\*\*\*\* Deadline \*\*\***

***Make the package/template CircularQue.ads into a generic with respect to queue size*.** Determine the minimum queue size to maximize the amount of food product sold during TWO days. This problem requires your cognitive insight. You may vary the size of the queue, number of product generators and number of sales task. **Your submission must utilize I/O redirection using the command prompt.** Your report must include the number of food generators used, number of sales task used, the number of meat food packets sold, other food packets sold and total food packets sold. Submit the results of at least the first 2 hours of execution and 2 hours prior to the final results by copying the material from the file created by your I/O redirection. Paste the results into your report. You need not include the entire result file.

**“A: Option (maximum grade is 100):**

**You are only eligible to submit the “A” if you met the deadline for the “B“ option.**

**Impress me! Do something extra. You must clearly tell me what you did on your cover page and highlight the code utilized with a highlighter! Adding a useful function or procedure would be sufficient as long as you show me the code and how it was used.**

**Appendix A:**

**Source for executable system.**

--in file CircularQue

--

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

--\*\* This implementation of the circular queue sacrifices run-time allowing complete utillization of all queue space.

-- Message should not be sent without verifying the queue has avaliable storage space!!!!

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

generic

type message is private;

capacity: in Natural := 24; -- Default queue size

package CircularQue is

procedure acceptMessage(msg: in message);

procedure retrieveMessage(msg: out message);

function circularQueEmpty return Boolean;

function circularQueFull return Boolean;

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

end CircularQue;

**-----------------------------**

-- in file Circular\_Que

package body CircularQue is

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

use IntIO;

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

front, rear: slotindex := 0; -- insert at front, remove from rear.

mesnum : Natural range 0..(capacity-1) := 0; -- number in buff

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

maxMessages: Natural := capacity - 1; -- Integers >= 0.

procedure acceptMessage(msg: in message) is

begin

if mesnum < maxMessages then -- reserve space and insert msg.

rear := (rear + 1) mod capacity; -- implement wrap-around.

box( rear ) := msg;

mesnum := mesnum + 1;

else

put("ERROR - Message rejected - queue is full!"); new\_line(2);

end if;

end acceptMessage;

procedure retrieveMessage(msg: out message) is

begin

if mesnum > 0 then -- remove message if buff not empty

front := (front + 1) mod capacity; -- front trails the next message by 1. rear is the actual last msg.

msg := box(front);

mesnum := mesnum - 1;

else

put("ERROR - No message in the queue to retrieve!"); new\_line(2);

end if;

end retrieveMessage;

function CircularQueEmpty return Boolean is

begin

if mesnum > 0 then

return False;

else

return True;

end if;

end CircularQueEmpty;

function CircularQueFull return Boolean is

begin

if mesnum < maxMessages then

return False;

else

return True;

end if;

end CircularQueFull;

end CircularQue;

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

-- file Distribution\_Service.ads;

--

-- The software suite consists of Product Distribution Main,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service moduke will be discarded once the embedded software require to

-- manage the physical system is complete and installed.

--

-- This package simulates the arrival of food packets for the GateKeeper Service storage facility.

-- Food\_Packs are removed from interplanetary shipping crates by the GateKeeper

-- and repacked into smaller Food\_Packs more appropriate for handling at their final destination

-- planet. The GateKeepr provides the new Food\_Packs Food\_SalesService points-of-sale for

-- disbursement.

--

-- This task will be discarded in the final implementation as real cargo vessels will contact the

-- GateKeeper directly to take their loads.

with Food\_DataStructures; use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

package Distribution\_Service is

task type Product\_Generator;

end Distribution\_Service;

-- in file Distribution\_Service.adb;

--with Food\_DataStructures; use Food\_DataStructures;

--with Stats\_FoodDistribution; use Stats\_FoodDistribution;

with Food\_DataStructures; use Food\_DataStructures;

with GateKeeperService; use GateKeeperService;

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

with Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

use Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

package body Distribution\_Service is

--package MyGateKeeper is new GateKeeperService( 4 ); use MyGateKeeper;

task body Product\_Generator is -- task body

RandFloatGenerator : Generator; -- Has own generator default seed.

randFloat : Float;

newFood: Food\_Pack;

nextArrivalDelay: float;

begin

-- Start the simulation with an arriving vessel.

loop

--put("In Product\_Generator of DistributionServices."); new\_line(2);

setFood\_PackFoodType( newFood, Food\_DataStructures.RandomFoodType );

if getFood\_PackFoodType(newFood) in GrainVegetable then

setFood\_PackShipment( newFood, 'B'); -- 'B' for basic Grain or vegetable.

else

setFood\_PackShipment( newFood, 'M'); -- 'M' for meat.

end if;

if getFood\_PackFoodType( newFood) in GrainVegetable then

delay( PrepareGrainVegetableFood\_PackforSales );

else

delay( PrepareMeatFood\_PackforSales );

end if;

PrintFood\_PackShipment( newFood ); put(" delivered. "); new\_line(2);

-- Blocking request for service. We are placed at the rear of the queue "Insert"

-- associated with the task "GateKeeper" and moved from the run state to the waits

-- state by the OS. We will not be eligible for execution again until "Insert" exits

-- the rendezvous.

GateKeeper.acceptMessage( newFood );

-- Schedule arrival of next star ship exponentially distributed over 1.534 seconds (hours).

-- Time in Ada is of type "duration" defined in package Calendar.

-- Cast/coerce floating calcultion "Random(RandFloatGenerator)\* 3.534" to duration.

nextArrivalDelay := Next\_Exponential; --Put("\*\*\*\* Next exp "); put(nextArrivalDelay);

nextArrivalDelay := nextArrivalDelay \* 1.534; -- Food cargo vessels arrive every 1.534 hours

Put("Next grain shipment arrives "); put(nextArrivalDelay); Put(" Time units!"); -- exponentially distributed.

new\_line(2);

-- Schedule the random arrival of the next cargo ship exponentially distributed (Poission Distribution).

-- Sleep till next ship arrives by casting float to duration.

delay Standard.Duration( nextArrivalDelay) ;

end loop;

end Product\_Generator;

end Distribution\_Service;

-- In file Food\_DataStructures.ads

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

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

with Ada.Numerics.Discrete\_Random;

package Food\_DataStructures is

--

-- Use 1.0 seconds of real time to simulate passage of 1 hour simulated time.

--

type Food\_Type is (Wheat, Corn, Rice, Potatoes, Squash, Tomato, Steak, Pork, Fish, Fowel);

subtype GrainVegetable is Food\_Type range Wheat .. Tomato;

package Food\_TypeIO is new Ada.Text\_IO.Enumeration\_IO(Food\_Type);

use Food\_TypeIO;

type Food\_Pack is private;

procedure PrintFood\_Pack( foodIn: in Food\_Pack );

procedure PrintFood\_PackType( FoodIn: in Food\_Pack );

procedure PrintFood\_PackShipment(FoodIn: in Food\_Pack);

function getFood\_PackFoodType( FoodIn: in Food\_Pack ) return Food\_Type;

function getFood\_PackFoodShipment( FoodIn: in Food\_Pack ) return Character;

procedure setFood\_PackFoodType( FoodIn: in out Food\_Pack; FoodType: Food\_Type );

procedure setFood\_PackShipment( FoodIn: in out Food\_Pack; FoodShipment: Character );

function RandomFoodType return Food\_Type;

private

type Food\_Pack is record

aFoodType: Food\_Type := Fowel;

aFoodShipment: Character := 'M';

end record;

end Food\_DataStructures;

--

-- in file Food\_DatatStructures.adb

--

package body Food\_DataStructures is

package GenerateRandomFoodType is new Ada.Numerics.Discrete\_Random( Food\_Type );

use GenerateRandomFoodType;

G: GenerateRandomFoodType.Generator; -- uniformly distributed

procedure PrintFood\_Pack( foodIn: in Food\_Pack ) is

begin put(FoodIn.aFoodType); put(" "); put(FoodIn.aFoodShipment); end PrintFood\_Pack;

procedure PrintFood\_PackType( FoodIn: in Food\_Pack ) is

begin put( FoodIn.aFoodType); end PrintFood\_PackType;

procedure PrintFood\_PackShipment(FoodIn: in Food\_Pack) is

begin put( FoodIn.aFoodShipment ); end PrintFood\_PackShipment;

function getFood\_PackFoodType( FoodIn: in Food\_Pack ) return Food\_Type is

begin return FoodIn.aFoodType; end getFood\_PackFoodType;

function getFood\_PackFoodShipment( FoodIn: in Food\_Pack ) return Character is

begin return FoodIn.aFoodShipment; end getFood\_PackFoodShipment;

procedure setFood\_PackFoodType( FoodIn: in out Food\_Pack; FoodType: Food\_Type ) is

begin FoodIn.aFoodType := FoodType; end setFood\_PackFoodType;

procedure setFood\_PackShipment( FoodIn: in out Food\_Pack; FoodShipment: Character ) is

begin FoodIn.aFoodShipment := FoodShipment; end setFood\_PackShipment;

function RandomFoodType return Food\_Type is

aFood: Food\_Type;

begin

aFood := GenerateRandomFoodType.Random( G ) ;

return aFood;

end RandomFoodType;

end Food\_DataStructures;

-- in file Food\_SalesService.ads

--

-- The software suite consists of Product Distribution Main,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- The specification require the ability to create multiple points of sale.

--

with Food\_DataStructures; --use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

package Food\_SalesService is

task type RetailSales;

--task RetailSales; -- May used if only a single point of sales is required.

end Food\_SalesService;

-- in file Food\_SalesService.adb

with Food\_DataStructures; use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

with GateKeeperService; use GateKeeperService;

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

package body Food\_SalesService is

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

use Integer\_IO;

task body RetailSales is

food: Food\_Pack;

availableForSale: Boolean := true;

begin

delay 1.0; -- Allow for initialization activities.

loop

GateKeeper.retrieveMessage( food, availableForSale );

-- The time to sell a product is exponentially distributed with mean 2.0 hours.

delay( duration( Next\_Exponential \* 2.0 ) );

put("Retail Sales successfuly sold "); PrintFood\_Pack( food ); new\_line;

end loop;

end RetailSales;

end Food\_SalesService;

-- in file GateKeeperService.ads

--

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite

-- represents the software to manage an "embedded" planetary system

-- food receiving and distribution system. The Distribution\_Service

-- module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

-- The GagteKeeper accepts Food\_Packs frfom inter-galatic transports

-- and repacks them into Food\_Packs suitable for Food\_SalesService to

-- distribute to the planets in the system. Food\_Packs are removed

-- from the default FIFO queue associated with the "accept" statement

-- and stored in a circular queue for distribution.

with Food\_DataStructures; use Food\_DataStructures;

With Stats\_FoodDistribution; use Stats\_FoodDistribution;

with CircularQue;

package GateKeeperService is

task GateKeeper is

-- accept Food\_Packs from interplanetary vesssels.

entry acceptMessage( newFood: in Food\_Pack );

--Allow sals to retrive the repackaged Food\_Packs.

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

end GateKeeper;

end GateKeeperService;

-- in file GateKeeperService.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 CircularQueue is new CircularQue (Food\_Pack, 10 ); -- default size 10.

use CircularQueue;

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

-- Terminate after losing 5 customers or time to close has arrived.

while rejected < 5 and Ada.Calendar.Clock < End\_Time loop -- Terminate after losing 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. Wakes when any “accept” is requested.

-- 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 not( CircularQueFull ) then

CircularQueue.acceptMessage( newFood);

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(CircularQueue.circularQueEmpty) then

availableForShipment := True;

CircularQueue.retrieveMessage( newFood );

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

end if;

end retrieveMessage;

end select;

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

end loop;

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

-- and number of arriving food vessels rejected.

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;

-- in file Stats\_FoodDistribution.ads

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

--

-- The software suite consists of ProductDistributionMain,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service, CircularQue and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service module will be discarded once the embedded software required to

-- manage the physical system is complete and installed.

--

with Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

use Ada.Numerics.Float\_Random, Ada.Text\_IO, Ada.Float\_Text\_IO, Ada.Integer\_Text\_IO;

with Food\_DataStructures; use Food\_DataStructures;

package Stats\_FoodDistribution is

--

-- Use 1.0 seconds of real time to simulate passage of 1 hour simulated time.

--

my\_Generator: Generator;

-- Exponential distribution using interpolation and famous data points from early IBM

-- Fortran statistical packages.

function Next\_Exponential return float;

-- Time requried to arrange raw food packets for sale.

function PrepareGrainVegetableFood\_PackforSales return duration;

function PrepareMeatFood\_PackforSales return duration;

end Stats\_FoodDistribution;

--

-- in file Stats\_FoodDistributionAdb.adb

--

--%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-- Generate exponentially distributed random deviates 0.0 <= x <= 1.0.

-- The function is used to generate random arrivals at the repair

-- facility by "Ship\_Generator" multiplied by the average arrival time.

--

-- The data points used to generate the deviates became famous

-- due to their use in IBM's FORTRAN compilers in the 1960's.

--

-- The exponential distribution is used to predict random events (Poisson)

-- as the probability of the next event occurring increases the longer it

-- gets since the last event occured. Examples: a) A car arriving at

-- an intersection. b) A shopper entering a store.

--

package body Stats\_FoodDistribution is -- Step function

-- 33.3% of grain/vegetable food packs require 0.25 hours preparation for sales.

-- 56.7% of grain/vegetable food packs require 0.66 hours preparation for sales.

-- 10% of food packs are damaged and require 0.75 hours preparations for sales.

--

function PrepareGrainVegetableFood\_PackforSales return duration is -- Step function.

randFloat: Float;

begin

randFloat := Random(my\_Generator); -- 0.0 <= randFloat <= 1.0.

if randFloat <= 0.333 then

return duration(0.25);

else if randFloat <= 0.9 then

return duration(0.66);

else

return duration(0.75); -- broken pallet of food.

end if;

end if;

end PrepareGrainVegetableFood\_PackforSales;

--

-- 33.3% of meat food packs require 0.25 hours preparation for sales.

-- 53.4% of meat food packs require 0.4 hours preparation for sales.

-- 13.3% of food packs are damaged and require 1.0 hours preparations for sales.

--

function PrepareMeatFood\_PackforSales return duration is -- Step function.

randFloat: Float;

begin

randFloat := Random(my\_Generator); -- 0.0 <= randFloat <= 1.0.

if randFloat <= 0.333 then

return duration(0.25); --

elsif randFloat <= 0.867 then

return duration(0.4);

else

return duration( 1.0 ); -- damaged shipment of meat

end if;

end PrepareMeatFood\_PackforSales;

--&&&&&&&&&& Plot of points (Xj, Yj) for j in range 0.0. to 1.0. with interpolation.

function Next\_Exponential return float is

x: float;

begin

x:= Random(My\_Generator); -- Return next exponential arrival interval.

if x = 0.0 then -- Mean arrival time = 1.

return 0.0;

elsif x <= 0.1 then

return ((x - 0.0) \* 1.04 + 0.0);

elsif x <= 0.2 then

return ((x - 0.1) \* 1.18 + 0.104);

elsif x <= 0.3 then

return ((x - 0.2) \* 1.33 + 0.222);

elsif x <= 0.4 then

return ((x - 0.3) \* 1.54 + 0.355);

elsif x <= 0.5 then

return ((x - 0.4) \* 1.81 + 0.509);

elsif x <= 0.6 then

return ((x - 0.5) \* 2.25 + 0.690);

elsif x <= 0.7 then

return ((x - 0.6) \* 2.85 + 0.915);

elsif x <= 0.75 then

return ((x - 0.70) \* 3.60 + 1.2);

elsif x <= 0.8 then

return ((x - 0.75) \* 4.40 + 1.38);

elsif x <= 0.84 then

return ((x - 0.8) \* 5.75 + 1.60);

elsif x <= 0.88 then

return ((x - 0.84) \* 7.25 + 1.83);

elsif x <= 0.9 then

return ((x - 0.88) \* 9.00 + 2.12);

elsif x <= 0.92 then

return ((x - 0.90) \* 11.0 + 2.30);

elsif x <= 0.94 then

return ((x - 0.92) \* 14.5 + 2.52);

elsif x<= 0.95 then

return ((x - 0.94) \* 18.0 + 2.81);

elsif x <= 0.97 then

return ((x - 0.95) \* 30.0 + 2.99);

elsif x <= 0.97 then

return ((x - 0.96) \* 30.0 + 3.20);

elsif x <= 0.98 then

return ((x - 0.97) \* 40.0 + 3.50);

elsif x <= 0.99 then

return ((x - 0.98) \* 70.0 + 3.90);

elsif x <= 0.995 then

return ((x - 0.99) \* 140.0 + 4.60);

elsif x <= 0.998 then

return ((x - 0.995) \* 300.0 + 5.30);

elsif x <= 0.999 then

return ((x - 0.998) \* 800.0 + 6.20);

else

return ((x - 0.9997) \* 1000.0 + 8.0);

end if;

end Next\_Exponential;

end Stats\_FoodDistribution;

-- in file ProductDistributionMain.adb

-- The software suite consists of Product Distribution Main,

-- Food\_DataStructures, Stats\_FoodDistribution, GateKeeperService,

-- Distribution\_Service and Food\_SalesService.

--

-- With the exception of ProductDistributionMain this suite represents the software

-- to manage an "embedded" planetary system food receiving and distribution system.

-- The Distribution\_Service moduke will be discarded once the embedded software require to

-- manage the physical system is complete and installed.

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

with Food\_DataStructures; use Food\_DataStructures;

with Stats\_FoodDistribution; use Stats\_FoodDistribution;

with Distribution\_Service; use Distribution\_Service;

with Food\_SalesService; use Food\_SalesService;

with GateKeeperService; use GateKeeperService;

procedure ProductDistributionMain is

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

numProductGenerators: Positive := 1; -- number product generators.

numPOS: Positive := 2; -- number points of sale.

--SalesPerson: RetailSales; -- single sales center.

begin --body ProductDistributionMain

put("How many Product Generators? "); get(numProductGenerators); -- number receiving stations.

new\_line;

put("How many points of sale? "); get(numPOS); -- number receiving stations.

new\_line(2);

declare

FarmProducts: array(1..numProductGenerators) of Product\_Generator;

POS: array(1..numPOS) of RetailSales;

begin

null;

end;

end ProductDistributionMain;

**Appendix B:**

**Additional help if desired!**

**Hint 1:**

Assume the name of your program is Pgm1.exe. If you type "Pgm1" at the DOS prompt, input is normally expected from the keyboard and results are printed on the terminal. The command "Pgm1 > file1" would expect all input to come from the keyboard but the results would be routed to a disk file in the current directory named "file1." The command "Pgm1 < raw1 > results" would obtain input (as stream IO) from the file "raw1" and place the output in the text file "results" in the current directory.

**Hint 2:**

Normally Ada will not allow an integer to be assigned to a character variable as bits must be truncated. Similarly, Ada will not normally allow a character to be assigned to an integer as there are not enough bits (padding must occur). In both cases, Ada will flag a probable error at compile time letting the programmer know they have probably made a logic error. To tell the compiler you really mean to perform the indicated operation, the generic package “Unchecked\_Conversion” should be instantiated to allow the conversion as shown below. There is no actual runtime penalty for most instantiations of Unchecked\_Conversion (no run time function call) resulting in CPU overhead when using a certified Ada compiler. The generic conversion is simply allowed at compile time as in the example below.

with Ada.Text\_IO; use Ada.Text\_IO; -- read and write characters.

with Unchecked\_Conversion; -- standard package with every validated Ada translator

procedure ConvertCharacterInteger is

-- To read and write 16 bit integers on the PC compiler.

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

use MyInt\_IO;

-- instantiations to convert between integer and character formats, 16 bits versus 8 bits.

function integerToCharacter is new Unchecked\_Conversion(Integer, Character);

function characterToInteger is new Unchecked\_Conversion(Character, Integer);

c1, c2: Character;

int1, int2: Integer;

begin

c1 := 'A';

-- int1 := c1; --error in Ada, strongly typed, suspects programmer error.

int1 := characterToInteger( c1 ); -- Signal compiler to allow conversion.

put(" int1 = "); put(int1,4); put(" ");

put("c1 = "); put(c1); new\_line(2);

int2 := 66;

-- c2 := int2; -- error

c2 := integerToCharacter(int2);

put(" int2 = "); put(int2,4); put(" ");

put("c2 = "); put(c2); new\_line(2);

end ConvertCharacterInteger;

**Hint 3: To create vehicle names by appending a character. This demonstrates some character manipulation facilities. You should not require this in the lab.**

Remember a “slice” in Ada allows the user to treat a substring as a single unit. Given str has the value “ABCD” str(2..3) refers to the characters in positions 2 through 3 ) as a group from the string str, i.e., ”BC.” “str(2..2) would be the second character in the string.

Example:

S: string(1..10);

S(1..4) := “Bara”

S(4..7) := S(1..4);

Result: “S” is the string “BarBara”

**Hint 4: Using “enumeration types.” Should not need unless you get fancy.**

subtype LastCharOfName is Character range ‘A’ .. ‘Z’;

LastCharName: LastCharOfName;

LastCharName := ‘A’; -- or LastCharName := LastCharOfName’FIRST;

LastCharName := LastCharOfName’SUCC( LastCharName ); Sets LastCharName to ‘B’.

LastCharName := LastCharOfName’LAST; -- sets LastCharName to ‘Z’.

**Hint 5:**

**In application programming, it is occasionally desirable to treat the same unit of memory at different times as a different data type!**

Frequently used conversion trick in assembly, “C,” C++ and other languages.

ASCII Conversions

|  |  |  |  |
| --- | --- | --- | --- |
| Character | Decimal | Integer | Integer (32 bits) Binary |
| ‘0’ | 48 | 0 | 0-0000 |
| ‘1’ | 49 | 1 | 0-0001 |
| ‘2’ | 50 | 2 | 0-0001 |
| ‘3’ | 51 | 3 | 0-0011 |
| ‘4’ | 52 | 4 | 0-0100 |
| ‘5’ | 53 | 5 | 0-0101 |
| ‘6’ | 54 | 6 | 0-0110 |
| ‘7’ | 55 | 7 | 0-0111 |
| ‘8’ | 56 | 8 | 0-1000 |
| ‘9’ | 57 | 9 | 0-1001 |

Assume we wish to convert a 32 bit one digit integer to an 8 bit ASCII character. This may be accomplished by adding the character ‘0’ (48 decimal) to the integer and dropping the leading 24 zeros. As an example the 32 bit integer 3 may be converted to a character by adding the character ‘0’ (or decimal 48) then truncating the leading 24 bits.

Ex: 3 + 48 = 3 + ‘0’ = 51 => ’3’ in ASCII

Alternately an ASCII digit represented as character may be converted to an integer by subtracting the character ‘0’ (or 48) and padding the 24 bits to the left with zeros.

Ex: ‘3’ – 48 = 51 – 48 = 3 => 3 in base 10 or 11 in binary.

Using “C:”

ch: char = ‘3’;

int1: int = 0;

int1 = (int)ch - 48; //first coerce/cast the character to an integer then convert to integer 3.

Or alternately:

int1 = int(ch) – 48; //function form of casting.

ch = char(3 + 48); // yields the character 3.

Or

ch = (char) (3 + 48); // yields the character 3.

Simple clock maniopulation. Ads actually provides separate clocks to measure elapsed wall clock time and time actually spent by a process alternating execution with other states in the system.

-- See package Ada.Calendar and package Ada.Real\_Time for convenient functions

-- to manipulate time.

with Ada.Text\_IO;

with Ada.Calendar; use ada.calendar;

procedure TimeLoop is

Start\_Time : Ada.Calendar.Time;

End\_Time: Ada.Calendar.Time;

begin

Start\_Time := Ada.Calendar.Clock;

End\_Time := Start\_Time + 40.0;

while Ada.Calendar.Clock < End\_Time loop

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

delay 10.0;

end loop;

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

Ada.Text\_IO.Put\_Line("End of program");

end TimeLoop;

0.000000000

10.011000000

20.011000000

30.013000000

40.015000000

End of program

Notice the **“time drift”** in each execution of the loop due to system overhear and competition with other task in the operating system (clock, hard disk, communications software, internet connections) and other users.