Data Structures Lab 1 - Lists Spring 2021 Burris

Due prior to the start of class Friday February 26. The due date contains more time than required to complete the lab. Do no procrastinate! In general sickness, social problems etcetera will not be accepted as excuses to submit the lab late! Additional labs will be assigned prior to the due date.

Submission: For all options, the results should appear first followed by the complete code! You must convenience me you used I/O redirection for all input and output. A lab may not be submitted more than 4 times!

Most modern programming languages offer a “container” class for the convenience of users. Containers are typically inefficient with respect to CPU time and memory allocation compared to static native types. They may however offer great convenience for the developer. Commonly offered container classes include one for constrained objects (homogeneous) and a second for unconstrained objects (heterogeneous). Most container classes are further refined to incorporate containers termed “sequence” and “associative.” Sequence classes hold sequences (lists) of related items. Associative classes (dictionaries) associate a key with each element of the class then manipulate elements based on the keys. You may not utilize container classes in any language for any lab during the semester unless the lab specifically states you may use containers. Ada provides the following containers for “defined/constrained” classes:

1. Ada.Containers.Vectors
2. Ada.Containers.Doubly\_lLinked\_Lists
3. Ada.Containers.Hashed\_Maps
4. Ada.Containers.Ordered\_Maps
5. Ada.Containers.Hashed\_Sets
6. Ada.Containers.Ordered\_Sets

A similar group of classes/packages is provided for undefined/unconstrained types using similar names including Ada.Containers.Indefinite\_Vectors with specialized generic procedures such as Ada.Containers.Generic\_Array\_Sort and Ada.Containers.Generic\_Constrained\_Sort. The purpose of this lab is to develop the basic technology to implement (program) these types of capabilities in languages including Ada, C++, Java, Python, Groovy, Ruby and Smalltalk using more basic constructs. In essence you will extend the languages ability to implement some problems. Implementation of containers, complex numbers, etcetera using code placed in libraries extends the capabilities of languages. This technique is widely used to extend programming language capabilities and convenience for modern languages.

*The above containers may be of great use. They are however generic in the sense they try to be all things for all application areas and developers. Professional software engineers/programmers should expect to be tasked with creating mission specific software/containers with the above characteristics but tailored to individual business needs with extensions. This first lab will prepare us for that demand. Please do the “A” option*. You will need the technology from the “A” option in future labs to meet grading standards. You may use Ada, C++ or Java for all labs this semester. This course will emphasize compiled languages for systems/problems with “hard real time constraints.” In these systems hardware efficiency is typically more important than programmer convenience/efficiency. An example would be software used to land a passenger jet during less than ideal conditions. While you may use Java for all labs it will typically limit your grade to a “B” as it will not be able to meet all problem specifications. Ada and C++ are compiled languages. Java is interpretive. Interpretive languages frequently execute 10 to 50 times slower than compiled solutions for the same problem.

**“C” Option: Homogeneous (maximum grade is 70):**

Dear Ima Programmer,

We have been hired by NASA to create mission specific software for the trip to Mars. If you do not feel you are up to the task, please update your resume. On termination, you will be provided the opportunity to travel and find new opportunities at your own expense. Our initial requirements with syntax/semantic examples follow. We will extend these requirements in future labs. YOU MUST USE I/O REDIRECTION FOR ALL I/O PLACING RESULTS IN A FILE!

Initial requirements specify fixed size sequentially allocated list structures (arrays) to hold integer values. Sample list declarations and functionality follow:

|  |
| --- |
| CreateList – implement as generic/template package/class (A and B options) |
| AddToList( <value> ) |
| PrintList( ) --empty parenthesis not used in Ada |
| PrintList( <pt> ) |
| DeleteList( <pt> ) |
| ListLength( ) |

“CreateList( <listName>, <size>, <type> )” creates a list named <listName> with room for <size> items of the specified <type>. Items in the list are indexed from 1 to N. Storage must be allocated in stack memory, not the heap. Use of heap memory (Java) will result in a 10 point penalty. *Hints: Dynamic storage allocation at runtime using generics ( control of data type, values, etcetera) in Ada is explained on pages 52-59 of “Data Structures Program Class Notes” Sample code.“ “box1.ada,” page 58 of the Program Class Notes exhibits an example for dynamically allocating storage for arrays/lists in the stack at runtime.*

“AddToList( <value> )” adds the indicated <value> as the new **last** item in the list <listName> for the “C” option.

“PrintList( )” prints the entire contents of the list from front to end.

“PrintList( <pt> )” prints only the item at position <pt>.

“DeleteList( <pt> )” deletes the item at position <pt> from the list. The algorithm required to delete a random node efficiently from a list is located on about page 27 of the “Data Structures Class Notes.”

“integer ListLength( )” returns the number of items in the list (indexed 1..N). A return of zero indicates the list is currently empty.

Algorithms for insertion as last item in an array/list, in sorted order, and random deletion appear on pages 27-28 [Basic Sequentially Allocated List Operations] of the “Data Structures Class Notes.”

Samples:

pt, length: Integer;

inList: array (1..20) of Integer; -- Create list, see box1.ada for the “C” option.

-- MakeList( intList, 20, Integer); -- Create list, see box1.ada using generics for A/B option.

PrintList( ); -- [ null ]

AddToList( 12 );

AddToList( 3 );

AddToList( 7 );

PrintList( ); -- [12, 3, 7]. Note items appear in the order placed in the list.

length := ListLength( );

for pt in 1..length loop

PrintList( pt ); -- produces [12, 3, 7].

end loop;

Delete( 2 );

PrintList( ); -- [12, 7].

**“B” Option: Homogeneous (maximum grade is 85):**

Dear Ima Programmer,

We have discovered the need to use additional intrinsic data types such as Float and Long\_Integer. Please convert the “C” option code to generics (Ada) or templates (C++/Java). Create list via generic instantiation. To demonstrate your code works, process the “C” options transactions followed by the floating point transactions below. Your program **must** use the same physical stack space for both the “C” and “B” option lists/arrays. If you do not feel you are up to the task, please expect a substantial salary reduction or update your resume and leave the company (preferred). YOU MUST USE I/O REDIRECTION FOR ALL I/O PLACING THE RESULTS IN A FILE! If you implement the “B” option use your generic code to create the list of integers when processing the “C” option transactions.

You need not explicitly implement the “C” option. Your program should first create and process the “C” data. Reclaim the “C” option memory from the stack. Now create the list for the “B” option and process the following transactions USING THE SAME MEMORY SPACE used for the “C” option (see coding hint at end of assignment).

length: Integer;

-- Create homogeneous array of floats [array(1..6)].

package BList is new CreateList( 3, float ); use BList;

PrintList( ); -- [ null ]

AddToList( 12.7 );

AddToList( 3.5 );

AddToList( 7.6 );

AddToList( 9.6 ); -- [ failure, overflow ]

PrintList( ); -- [ 12.7, 3.5, 7.6 ]. Note items appear in the order placed in the list.

length := ListLength( );

for pt in 1.. length loop

PrintList( pt ); -- produces [12.7, 3.5, 7.6].

end loop;

Delete( 2 );

PrintList( ); -- [12.7, 7.6].

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

Dear Ima Programmer,

We have discovered the need to use programmer defined enumeration types. Please convert the “B” option code to generics (Ada) or templates (C++/Java). Create list via generic instantiation. To demonstrate your code works, process the “C” and “B” options transactions followed by the enumeration transactions below. You need not explicitly write the “C” and “B” option code. Rather use your “A” option code to create all lists using the same physical stack space for the “C,” “B” and “A” option lists/arrays. If you do not feel you are up to the task, please expect a salary reduction. YOU MUST USE I/O REDIRECTION FOR ALL I/O AND PLACING THE RESULTS IN A FILE! **You must use programmer defined enumeration types.**

The “AddToList( <value> )” for the “A” option must place new entries in the list in lexicographic (ascending sorted) order! The required algorithm may be found in the “Data Structures Class Notes” on approximately page 28.

Management expects to view the “C” option results followed by the “B” option results and finally the “A” Option results.

**“A” Option Transactions**

type JobType is (Programmer, Software\_Engineer, Sales, Inventory\_Control, customer, manager);

pt, length: Integer;

Package AList is new MakeList( jobList, 20, Job\_Type); -- generics (box1.ada)

PrintList( ); -- [ null ]

AddToList( Programmer );

AddToList( Software\_Engineer);

AddToList( Software\_Engineer);

AddToList( Sales);

AddToList( customer);

AddToList( Programmer);

PrintList( ); -- [ Programmer, Software\_Engineer, Software\_Engineer, Sales, customer,

-- Programmer]. *I am not concerned with upper/lower case output at pesent*.

length :=ListLength( );

for pt in 1.. length loop

PrintList( pt ); -- produces [12, 3, 7].

end loop;

Delete( 4 );

PrintList( ); -- [ Programmer, Software\_Engineer, Software\_Engineer, Sales, customer, Programmer].

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

HINTS (this code will compile and execute):

generic -- in file CreateList.ads

size: integer;

type itemType is private;

package CreateList is -- export the following behavior.

procedure AddToList( listItem: in itemType );

procedure PrintList; -- Ada does allows but not require empty ( ).

procedure PrintList( pt: in integer );

procedure DeleteList( pt: in integer );

function ListLength return integer; -- Ada does allows but not require empty ( ).

End CreateList;

package body CreateList is -- in file CreateList.adb

len: Integer;

list: array(1..size) of itemType;

procedure AddToList( ListItem: in itemType ) is begin null; end AddToList;

procedure PrintList is begin null; end PrintList;

procedure PrintList( pt: in Integer) is begin null; end PrintList;

procedure DeleteList( pt: in Integer) is begin null; end DeleteList;

function ListLength return Integer is

begin

return len;

end ListLength;

end CreateList;

-- Lab 1 in fle Lab1.adb

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

with CreateList;

procedure Lab1 is -- in file Lab1.adb

-- declartions

--

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

size: Integer;

begin

put("enter size for 'C' option: ");

get(size);

declare

-- for "C" option declare list directly as in box1.adb.: array

Clist: Array(1..Size) of Integer; **-- Allocate in stack;**

begin

-- process transactions for "C" option using a loop to read and print results

null;

end;

**-- CList no longer in scope. Space popped/reclaimed automatically from stack.**

get(size); -- for "B" option

declare

-- use the same format for creating the "C" list if you sip the "C" lab option.

package BList is new CreateList( 3, float ); use BList;

floatNum: float;

begin

-- process transactions for "B" option.

-- loop -- for or while statement

AddToList(floatNum); -- or full object oriented notation

BList.AddToList(floatNum);

-- end loop

end;

**-- BList no longer in scope. Space popped/reclaimed automatically from stack.**

get(size); -- for "A" option

declare

type JobType is (Programmer, Software\_Engineer, Sales, Inventory\_Control, customer, manager);

package JobTypeIO is new Ada.Text\_IO.Enumeration\_IO(JobType); use JobTypeIO;

-- use the same format for creating the "C" list if you sip the "C" lab option.

package AList is new CreateList( 20, JobType ); use AList;

job: JobType;

begin

-- process transactions for "A" option.

-- loop -- for or while statement

AddToList( job ); -- or with full object oriented notation

AList.AddToList( job );

-- end loop

end;

**-- AList no longer in scope. Space popped/reclaimed automatically from stack.**

end Lab1;