Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Laboratory Exercise 5

Objectives

1. Practice in the use of packages.

2. Practice in the development and use of abstract data types.

3. Practice in the use of access types in forming a linked list

Submitting Your Work

Save your package files (the .ads and .adb file) in the zipped tar file WnnnnnnnnLab5.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 5 Submission** item on the course web site. You must submit your program by 3:00pm on Tuesday, February 28, 2011.

Your Task

Your task is to rewrite the specification and body of an Ada package book_collections from Lab Exercise 4, using a linked list, instead of an array of books. Your revised book_collections package must still work with the provided program bookstuff.adb.

Note: the Ada source code of book_pack.ads, book_pack.adb and bookstuff.adb is provided for your information. **You must not change any of those files. Your book_collections package must be written to work with the provided files.**

Package book_pack

This package defines the private data type book_type, operations that can be performed on instances of this data type and data types used in the definition of book_type. You will find all the details you need in the specification file book_pack.ads.

The file book_pack.adb is only provided for your information. It shows you how the book_type ADT is implemented.

Note: This package is provided for you. It has been tested and found to work completely correctly. Do not modify this file. Your task is to develop the package book_collections.

Procedure bookstuff

This procedure is provided in the file bookstuff.adb. The procedure uses the resources from packages book_pack and book_collections. It will not compile until you develop the specification for the book_collections package in a file book_collections.adb. It will not build an executable file until you have written and compiled the book_collections package body in a file book_collection.adb.

Procedure bookstuff reads in the details of several books from two data files whose names are given on the command line. The data files list1.txt and list2.txt are provided for this purpose. All the I/O and the command line is handled by procedure bookstuff.

For each data file, bookstuff adds the books to a book_collection and performs some operations on the books in that collection: change the price of some books, add some stock for some books and sell some books. It then displays the contents of the collection.

After creating and manipulating a separate book_collection for each input file, bookstuff then merges the two collections and displays the merged collection.

Note: All this capability of bookstuff is already written. It has been tested and found to work completely correctly. Do not modify this file. Your task is to develop package book_collections to provide the resources needed by bookstuff.

Package book_collections

This is the package that you need to develop for the lab exercise. It will need to use the resources in package book_pack. All the I/O and the command line is handled by procedure bookstuff. Your package book_collections should not perform any I/O, except for debugging purposes, which you should remove or comment out before submitting the files for the exercise.

This package must provide the book collection resources needed by procedure bookstuff. You could deduce those needs from studying the file bookstuff.adb, but to save you the time and trouble, here is a list of the required resources from package book_collections.

Private type book_collection	This must be a private type and must be implemented as a record which includes the following components: • A linked list of records, each containing a book_type component. • The actual size of the collection, initially zero.
Function collection	This function, used in Lab Exercise 4, is not needed for Lab Exercise 5.
Procedure change_price	Given parameters for a collection, the ISBN of a book, and a price, this procedure must search the collection for a book with that ISBN and, if found, change its price, otherwise raise the exception Book_not_found.

Procedure change_stock	Given parameters for a collection, the ISBN of a book, and a quantity of books (negative or positive), this procedure must search the collection for a book with that ISBN and, if found, add the quantity to its stock, otherwise raise the exception Book_not_found.
Function size	Given a parameter specifying a collection, this function returns the number of books in that collection.
Function stock_value	Given a parameter specifying a collection, this function returns the total dollar value of the books in that collection.
Procedure Add_book	Given parameters specifying a collection and a book, this procedure adds the book to the collection, provided there is room in the collection and the book is not already there. If the collection has already reached its limit, raise the Collection_full exception. If the book is already in the collection, raise the Duplicate_book exception.
Function Merge	Given parameters for two book collections, this function creates and returns a new collection which contains all the books found in either or both collections. If any book is found in both of the collections passed as parameters, only one entry is to be added to the new collection for that book, with its stock value equal to the sum of the stock values in the two merged collections and its price set to the minimum price for that book in the merged collections.
Function ToString	Given a parameter specifying a collection, this function must return a string which is the concatenation of the strings returned by book_pack.ToString for all the books in the collection.
Exceptions	Book_not_found, Collection_full, Duplicate_book.

Program Requirements

The package book_collections must provide correct implementations of all the resources listed above, so that procedure bookstuff can build and execute correctly.

Your package book_collections will be graded on correct functionality, as specified above, and conformance to the coding standards, described below.

What you must submit

You must submit the Ada source files (the .ads and .adb file) for the package. These files must be included in a zipped tar file.

Saving your files in a zipped tar file

You only submit .adb and .ads files. First you need to bundle them up into a single tar file.

Use the command:

```
tar -cf WnnnnnnnnLab5.tar *.adb *.ads
```

(where Wnnnnnnnn is your W-number).

The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

This will include every file in the current directory whose name ends with ".adb" or ".ads".

If you now use the command ls you should now see the file WnnnnnnnnLab5.tar in your directory.

If you use the command

```
tar -tf WnnnnnnnnLab5.tar
```

(where Wnnnnnnnn is your W-number), it will list the files within the tar file.

Now compress the tar file using the gzip program:

```
gzip WnnnnnnnnLab5.tar
```

By using the ls command again, you should see the file WnnnnnnnnLab5.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 5 Submission** link in the moodle web site.

Some students have reported problems in using gzip. If this happens, just submit the tar file.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2. Use comments at the start of the program to identify the purpose of the program, the author and the date written.

3. Use comments at the start of each procedure to describe the purpose of the procedure and the purpose of each parameter to the procedure.

4. Use comments at the start of each section of the program to explain what that part of the program does.

5. Use consistent indentation:

   - The declarations within a procedure must be indented from the Procedure and begin reserved words. The body of the procedure must be indented from the begin and end reserved words. Example of procedure indentation:

```
procedure DoStuff is
     Count : integer;
     Valid : boolean;
begin
     Count := 0;
     Valid := false;
end DoStuff;
```

   - The statements within the then-part, each elsif-part and the else-part of an if statement must be indented from the reserved words if, elsif, else and end.

```
if Count > 4 and not Valid then
     Result := 0;
     Valid := true:
elsif Count > 0 then
     result := 4;
else
     Valid := false;
end if;
```

   - The statements within a loop must be indented from the loop and end loop.

```
loop
     Count := Count + 1;
     Get (Number);
     exit when Number < Count;
end loop;
```

   - The exception handlers and statements within each exception handler must be indented.

```
begin
     ...  -- normal processing statements
exception
     when Exception1 =>
          Put_Line ("An error has occurred");
          Total := 0;
     when others =>
          Put_Line ("Something weird happened");
end;
```