Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Laboratory Exercise 6

Objectives

1. Practice in the development and use of generic procedures and functions.

2. Practice in the use of abstract data types.

Submitting Your Work

Save your files (the .ads and .adb files) in the zipped tar file WnnnnnnnnLab6.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 6 Submission** item on the course web site. You must submit your program by 3:00pm on Tuesday, March 6, 2012.

Your Task

Your task is to write a generic function to calculate the average of some numeric component of an array of book_type data items and to instantiate and use that function to calculate average price, stock and stock value of a list of books. You must also instantiate various versions of the generic sort procedure and use those procedures to sort the list of books.

Type book_type is defined in the package book_pack, also used in Lab 4 and Lab 5.

You are provided with an Ada file bookstats.adb which provides the input and output of book_type data and the skeleton of processing to be performed on a list of books. You must modify this program to instantiate versions of the generic average function.

You are to instantiate three versions of your generic average function:

- one to find the average price of an array of books,
- one to find the average stock of an array of books and
- another to find the average stock value of and array of books.

You are also provided with the generic sort procedure, discussed in lectures. You must further modify bookstats.adb to instantiate versions of this generic sort procedure. You are to instantiate four versions of the generic sort procedure:

- one to sort the books in ascending order of ISBN
- one to sort the books into ascending order of price
- one to sort the books into descending order of stock
- another to sort the books into descending order of stock value.

You must also modify bookstats.adb to call the average calculation functions and sort procedures at the locations indicated in the body of procedure bookstats.

Note: the Ada source code of book_pack.ads, book_pack.adb, gen_sort.ads and gen_sort.adb is provided for your information. **You must not change any of those files. You also must not change the input/output procedures in bookstats.adb.**

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.

Procedure bookstats

This procedure is provided in the file bookstats.adb. The procedure uses the resources from package book_pack. It will compile and run without any modification and, given the provided data file List1.txt, it will display the list of books 5 times, each time in the order in which the books are listed in the data file.

Once you have instantiated and called the sort procedures, each display of the books will be in a different order.

Currently, the program does not calculate any averages. Once you have written your generic average function and instantiated it for average price, stock and stock value, you can use those functions to calculate and then display the averages with procedure Put.

Program Requirements

Your generic average function and your modifications to bookstats.adb 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 generic average function and the modified file bookstats.adb. 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 for the generic average function and the modified bookstats.adb. First you need to bundle them up into a single tar file.

Use the command:

```
tar -cf WnnnnnnnnLab6.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 WnnnnnnnnLab6.tar in your directory.

If you use the command

```
tar -tf WnnnnnnnnLab6.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 WnnnnnnnnLab6.tar
```

By using the `ls` command again, you should see the file WnnnnnnnnLab6.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 4 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;
```