Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Laboratory Exercise 2

Objectives

1. Practice in the use of recursion.

2. Practice in the use of the command line parameters.

Submitting Your Work

Save your program in a zipped tar file, named WnnnnnnnnLab2.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 2 Submission** item on the course web site. You must submit your program by 3:00 pm Monday Feb 6 2012.

Your Task

Your task is to write an Ada program to solve the Towers of Hanoi puzzle multiple times with the number of discs in a range specified by the user. The program is to count the number of disc movements for each puzzle solution and give the user the option of having the disc movements listed on the monitor.

The range of numbers of discs is to be obtained from the command line. The program will then solve the Towers of Hanoi puzzle multiple times, once for each number of discs in the specified range.

The command line arguments are:

1. The minimum number of discs for the puzzle.adb

2. The maximum number of disks for the puzzle.

3. The character 'V' or 'v' (for "verbose"), indicating that the user wants a list of the disc movements.

None of the command-line arguments is required.

1. If there are no command line arguments, the program must solve the puzzle for 3 discs, report the number of disc movements, but not list the disc movements.

2. If there is just one command line argument it should be an integer number – if it is not an integer number, it is to be ignored and the value 3 used in its place. The program must solve the puzzle for this number of discs, report the number of disc movements, but not list the disc movements. adb

   Note: Each command line argument is a string. You can obtain the integer value of the string by using a form of the Get procedure from Ada.Integer_Text_IO.

   ```
   procedure Get(From: in String;     -- the string argument
                 Item: out Integer;   -- the integer value
                 Last: out Positive); -- the string length
   ```

   Procedure Get raises a Data_Error if the string cannot be translated into an integer.

3. If there are two command line arguments, they should both be integer numbers – if either command line arguments is not an integer number it is to be ignored and the value 3 used in its place. The program must solve the puzzle multiple times, starting with the number of discs set to the first number and increasing by one each time until it is greater than or equal to the second number. For each number of discs, the program is to report the number of disc movements, but not list the disc movements.

   For example, if the name of the program is hanoi:

   ```
   hanoi 3 6
   ```

   will solve the puzzle 4 times, for 3, 4, 5 and 6 discs, respectively.

   ```
   hanoi 3 2
   ```

   will solve the puzzle once, for 3 discs.

   ```
   hanoi dog 6
   ```

   will solve the puzzle 4 times, for 3, 4, 5 and 6 discs, respectively.

4. If there are three command line parameters, the first two must be integer numbers and are used as described above in point 3 (for two command line parameters). If the third command line parameter is 'v' or 'V', the program is to list the disc movements, one per line, for example:

   ```
   Move disc from peg 1 to peg 3
   ```

Your program 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 file (the .adb file). This file must be included in a zipped tar file.

Saving your files in a zipped tar file

Note: you will probably have just one Ada source file for this lab, so thses instructions may seem like an overkill, but this is the method you are to use for later labs and assignments, so treat this as practice!

You only submit .adb and .ads files. First you need to bundle them up into a single tar file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

Use the command:

```
tar -cf WnnnnnnnnLab2.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 WnnnnnnnnLab2.tar in your directory.

If you use the command

```
tar -tf WnnnnnnnnLab2.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 WnnnnnnnnLab2.tar
```

By using the `ls` command again, you should see the file WnnnnnnnnLab2.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 1 Submission** link in the moodle web site.

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;
```