COE428 Lab 2: Recursion

Program development and debugging

This is a one week lab. This lab must be submitted at least 48 hours before the beginning of your next lab period.

Prelab preparation

Before coming to the lab you should:

- Read the lab. Try to prepare any questions you may have about the lab.
- Refer to **Lab Guide**.
- Create your lab directory for lab2. (i.e. use **mkdir lab2** within your coe428 directory.)
- Change to your coe428/lab2 and unzip the lab2.zip file with the command:
  unzip /home/courses/coe428/lab2/lab2.zip

Introduction

This lab reviews even more basic C programming with an emphasis on recursive algorithms. You will:

- Use programs to aid your understanding of recursion.

Tutorial I: Recursion in C

You have learned in your lectures how to describe and code (in C) several algorithms that are recursive. We re-examine the classic *Towers of Hanoi* algorithm here.

Specifications for executable

The specs are:

- Towers are identified with a single integer: the left, middle and right towers are named as '1', '2' and '3' respectively.
- The program should print (to *stdout*) the moves required to solve the problem as a sequence of lines in the format:

```
FROM_ID SPACE DEST_ID
```

For example, to move 3 disks from Tower 1 to Tower 2, the *stdout* output should be:

```
1 2
1 3
2 3
1 2
3 1
3 2
1 2
```

- *NOTHING ELSE* should be written to *stdout*.

The algorithm can be expressed in C as:

```
#include <stdio.h>
#include "towers.h"
void towers(unsigned int n, unsigned int from, unsigned int dest)
{
    unsigned int spare = 6 - from - dest;
    if (n != 0) {
        towers(n-1, from, spare);
        printf("%d %d\n", from, dest);
        towers(n-1, spare, dest);
    }
}
```

Displaying debugging information

In order to see how the algorithm runs, it is useful to display additional information as the program runs. For example, more insight might be gained if the output was something like:

```
towers(3, 1, 2)          Initial invocation
..towers(2, 1, 3)        Recurse (indentation indicates depth)
....towers(1, 1, 2)      Recurse
......towers(0, 1, 3)    Recurse (base case, returns immediately)
......Move #1: From Tower 1 to Tower 2  First move (by towers(1, 1, 2)
......towers(0, 3, 2)
....Move #2: From Tower 1 to Tower 3
....towers(1, 2, 3)
......towers(0, 2, 1)
......Move #3: From Tower 2 to Tower 3
......towers(0, 1, 3)
..Move #4: From Tower 1 to Tower 2
..towers(2, 3, 2)
....towers(1, 3, 1)
......towers(0, 3, 2)
......Move #5: From Tower 3 to Tower 1
......towers(0, 2, 1)
....Move #6: From Tower 3 to Tower 2
....towers(1, 1, 2)
......towers(0, 1, 3)
......Move #7: From Tower 1 to Tower 2
......towers(0, 3, 2)
```

How to do this?

The specs require a precise format for the *stdout* output. Thus we are not allowed to print this additional information to *stdout*. We opt to use *stderr*.

The C code

The C source code to do this is shown here:

```
/* Author: kclowes */

/* Description: Solves "Towers of Hanoi" problem.

 *              Prints sequence of moves to stdout.

 *              Prints other information tracing the

 *              algorithm's progress to stderr.

 */

#include <stdio.h>

#include "towers.h"


static void showRecursionDepth(void);

static unsigned int depth = 0;

static unsigned int moveNumber = 0;


void towers(unsigned int n, unsigned int from, unsigned int dest)

{

    unsigned int spare = 6 - from - dest;

    showRecursionDepth();

    fprintf(stderr, "towers(%d, %d, %d)\n", n, from, dest);

    depth++;

    if (n != 0) {

        towers(n-1, from, spare);

        showRecursionDepth();

        fprintf(stderr, "Move #%d: From Tower %d to Tower %d\n",

                        ++moveNumber, from, dest);
```

```
        printf("%d %d\n", from, dest);
        towers(n-1, spare, dest);
    }
    depth--;
}

static void showRecursionDepth()
{
    int i;
    for(i = 0; i < depth; i++)
        fprintf(stderr, "..");
}
```

Compile and run towers

You do not need to type in any of the C source code files; you got copies of the necessary files when you copied the "needed" files for the lab.

Create the executable towers with the command:

```
gcc -o towers towers.c towersMain.c
```

You can now run the program with the command:

```
towers
```

Lots of lines of information will be displayed on the command line window. The next section explains how you can view what is of interest.

Separating stdout and stderr

We have seen previously that the *stdout* stream using the > symbol. Try it with the towers command as follows:

```
towers > junk1
```

You will still see the quite verbose output written to *stderr* but the output written to *stdout* will no longer appear on the screen (and mixed up with *stderr*); instead, it is put into the file junk1 (where you can view it at your leisure).

It is also possible to redirect *stderr* with the two-character symbol **2>**. For example, we can do the opposite of the previous example (seeing *stdout* on the screen and re-directing *stderr* to a file) with the command:

```
towers 2> junk2
```

You can also redirect both *stdout* and *stderr* at the same time; in this case, each will be written to a separate file and nothing will appear on the screen. Try the following command

```
towers 2> details > pureStdout
```

Requirement 1

Answer these questions in your README file.

Question:

Suppose that towers(5, 2, 3) is invoked.

1. How will the first recursive call to towers() be invoked? Answer this question in the form: towers(x, y, z) where you give the actual values to the three parameters.
2. How many recursive calls to towers() will be made before this first recursive call actually returns to the initial invocation?
3. Once towers(5, 2, 3) has invoked its first recursive call to towers() and this invocation has returned, what will be printed to *stdout*? (i.e. What actual move will be made by towers(5, 2, 3) ?)

4. How will the second recursive call to towers() be invoked? Answer this question in the form: towers(x, y, z) where you give the actual values to the three parameters.

Question:

Suppose that towers(8, 1, 2) is invoked. How many lines will be printed to *stdout*?

Note

- You should note (or try to convince yourself) that the number of lines printed to *stdout* is precisely equal to the number of moves required to solve the problem.
- You can use the theoretical analysis of the problem to determine the number of moves.
- The towers command behaves somewhat differently than what has been described so far. In particular, if it is invoked with an argument, it will move the specified number of disks from Tower 1 to Tower 2. For example,

```
towers 10
```

will output the actions to move 10 disks.

- Of course, you can redirect the moves to a file and then count the number of lines in the file, allowing you to use the software to verify your theoretical answer.
- Instead of counting the lines manually, you can use the Unix command wc -l someFile to do the work for you.

Tutorial II: The main() function

Specifications for the towers command

NAME

towers—towers of Hanoi solver

SYNOPSIS

towers *numberDisks fromTower destTower*

DESCRIPTION

Solves the *Towers of Hanoi* problem and prints the required moves to *stdout*. One line of text in the format FromTowerID ToTowerID is written for each move. The TowerIDs are '1' (for left tower), '2' (for middle tower) or '3' (for right Tower).

The program behaves as follows depending on the arguments given on the command line:

No arguments

> If no arguments are given, the problem is solved for moving 3 disks from Tower 1 to Tower 2.

One argument (*numberDisks*)

> If only one argument is given, the problem is solved for moving *numberDisks* disks from Tower 1 to Tower 2.

Three arguments (*numberDisks fromID toID*)

> If all three arguments are given, the problem is solved for moving *numberDisks* disks from Tower *fromID* to Tower *toID*. The tower IDs must be either '1', '2' or '3' and the two IDs must be different.

EXIT CODE

If the command is invoked correctly, the moves are output and the exit status is 0 (zero).

Otherwise, an incorrect invocation produces no output. A message is displayed on *stderr* and the exit status is non-zero.

BUGS

The program does not yet behave as specified. The command line args to identify the two towers are not recognized. Furthermore, incorrect invocation is not detected.

Initial version of main()

You are provided with an initial version of the main() function in the file towersMain.c and the listing is shown below. However, the bugs identified in the previous section can be fixed in the main() function.

```
#include <stdlib.h>

#include "towers.h"
```

```
int main(int argc, char **argv)

{

    int n = 3;

    int from = 1;

    int dest = 2;

    if (argc > 1) {

        n = atoi(argv[1]);

    }

    towers(n, from, dest);

    exit(0);

}
```

Requirement 2

You have to modify the main() function so that the bugs identified are fixed. The file containing the main function must still be calledtowersMain.c; i.e. you edit the existing file to fix the bugs.

Lab Submission

README file

Your README file should contain:

1. A brief description of what you did (and did not) achieve in the Lab.
2. Your answers to the Questions in Requirement 1.

Submit your lab

1. Go to your **coe428** directory
2. Zip your **lab2** directory by using the following command:
   zip -r lab2.zip lab2/
3. Submit the lab2.zip file using the following command:
   submit coe428 lab2 lab2.zip

by Ken Clowes, revised by Olivia Das