COMP2401—Tutorial 2
C functions and binary data representation

Learning Objectives

After this tutorial, you will:
* Refresh your memory about coding recursive functions
* Learn about the tar command
* Submit tar files to CULearn
* Practice binary data representation – pencil and paper

1 Coding a recursive function

The purpose of this section is to experiment with recursive functions. In this section of the tutorial you will use the file printTriangles.c. You can see the expected result by executing the file print_triangles_prog.

Recursive code consists of three parts:

- **Base Case** – this part consists of a set of conditions that terminates the recursive call. Note that the base case may consist of multiple statements.
- **Recursive Step** – this part of the recursion determines how the recursive step will be carried out.
- **Action** – this part of the recursion determines the side effect that is created by the function. At times the side effect may be combined with the recursive step and at times it may not

Note, that the order in which the parts Base Case, Recursive Step and Action appear in a recursive function may be different from function to function.

Computing n! (n factorial) is a classic example of a recursive function. For example, 4!=4*3*2*1, which is 24. Following the parts of the recursion stated above we have the following parts for computing n!

- **Base case** – when n is 0 then return 1. Note that 0! == 1
- **Recursive step** – the recursive step computes (n-1)!. This is because if one knows the answer to (n-1)! then all is left to do is to multiply the answer by n in order to determine the answer to n!.
- **Action** – The side effect that is created in this part is the multiplication of n by the answer to (n-1)!.

Thus the prototype of a function that computes n! is
unsigned int factorial(unsigned int num);
the pseudo code is as follows:
// Base case
when n is 0 then return 1;
// Action and Recursive steps combined
return n * factorial(n-1)

1.1 Task 1

In this task you are asked to print a right triangle using '*'. The input to the function is the number of rows. For example, if the number of rows is 5 then the output should be

```
*
**
***
****
*****
```

Function prototype (here you can assume that the input is always > 0).

`void printTriangleRecursive(int numRows);`

Steps:

1. Identify the three parts (base case, recursive step, and action)
2. Write the pseudo code for each part of the recursion
3. Code the function

Hints:

- Use a for loop or a while loop to print the * in each row using **printf** function (e.g., **printf("*");**)
- Use '\n' to skip a line (e.g., **printf("\n");**)

Solution

1. Identify the three parts (base case, recursive step, and action)
    a. Base case – check if numRows is 0
    b. Recursive step – reduce the number of row by one (1) and invoke the recursive call.
    c. Action – print the stars for the current row

2. Write the pseudo code for each part of the recursion

```
// base case
If (numRows == 0) return 0;
// recursive call
Call the function with numRows-1
//  action part
Using for loop  print numRows *
Skip a line
```

1.2 Task 2

Copy the function **`printTriangleRecursive()`** that you just coded and save it as a new function where the function prototype is

`void printTriangleUpsidedown(int numRows);`

The new function should print the triangle with the base at the top. For example, if the number of rows is 5 then the output should be

```
*****
****
***
**
*
```

Here, do not rewrite the function code. Instead you need to rearrange the order of some parts of the recursion. Determine which parts you need to reorder.

1.3 Task 3 (optional)

This is somewhat more challenging. Here you are required, in addition to printing the triangles, to compute and return the number of '*' that were printed. For example, if numRows is 10 then the parameter numStars should have the value of 55.
Note,

1. The number of stars that were printed is returned in the parameter numStars and not as a return value by the function.
2. One cannot assume that numStars was initialized to 0

The function prototype

```
void printTriangleRecursiveCountStars(int numRows, int *numStars);
```

Hints:

- Set numStars to 0 in the base case part
- Update numStars during the action part.

2 TAR Files

TAR files are files that were originally designed for archiving directories and creating backups on tapes. TAR stands for Tape Archive. In this course you will submit all your assignments using TAR files. In this part of the tutorial you will learn how to use TAR files.

1. From your home directory, create a new directory called 'Tutorial2'.
2. Create a **Readme** text file containing
   - your name and studentNumber
   - a list of source files to be submitted (printTriangles.c)
   - instructions for compiling and running your program
3. Move or copy printTriangles.c that you worked on into the 'Tutorial2' directory.
4. Files with the '.tar' extension represent **T**ape **Ar**chive files, which are single files produced by the concatenation of their archived files. As such, the contained files are not compressed. These tar files are a common way of collecting files together into an archive in Unix/Linux.

   The "tar" program manipulates tar files. As such it accepts command line options that instructs the program what to do

* To create a tar file – use –cvf to create a file where (-c is for creating the file, -v is for verbose and –f is for the target file name). The command is:

```
tar  -cvf   Tutorial2.tar  *
```

or

```
tar -cvf  Tutorial2.tar Readme printTriangles.c
```

* Listing what is in a tar:

```
tar -tvf Tutorial2.tar
```

* Extracting the files in a tar file – use –xvf (-x is for listing the content, -v is for verbose and –f is for specifying the target file name). The command is:

```
tar -xvf Tutorial2.tar
```

Additional options can be found in the manual page for the 'tar' program.

5. Submit the 'Tutorial2.tar' file on *cuLearn* under the Tutorial 2 section.

End of tutorial section for submission

3 Binary data representation (pencil and paper exercises)

Below are additional exercises that you can do to make sure that you understand the concept of binary data representation. You will find it helpful to review course notes Chapter 2 section 2.1 "Number Representation and Bit Models".

3.1.1 Given $1101\ 0111_2$ what is decimal, octal and hex representation?

3.1.2 What are the limits – X, Y and Z – in decimal of an 8-bit, 12-bit and 16-bit magnitude only binary numbers?

For example, 4-bit: $0\text{-}15_{10}$ 8-bit: $0\text{-}X_{10}$ 12-bit: $0\text{-}Y_{10}$ 16-bit: $0\text{-}Z_{10}$

3.1.3 **If we have a range from 0-359_{10} how many bits do we need to represent it?**

3.1.4 **Given an octal number 355_8 what are its binary, hex and decimal representations?**

3.1.5 **Given a decimal number 355_{10} what are its hex, octal and binary representations?**

3.1.6 **For the above question, what do you find easier to do: first convert to octal then to hex and binary, or first to binary then hex and octal, or … (your method)?**

3.1.7 **Fill in this table:**

Decimal	Binary	Octal	Hex
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			