COMP 206, Fall 2018, Assignment 2

Objective

To provide you with your first experience programming in **C**. First, we'll try to give you a solid comfort level with the language by implementing some given algorithms that we describe (these may contain new algorithmic concepts such as recursion, which we hope you'll learn also). Then, we will start to work with string data by parsing real-world data from an important page on the internet, focusing on the nature of real text documents and looking at C's string processing functionality.

Getting Started

There are no provided files this time, but rather you will start from scratch writing C code. We recommend that you create your solution files within the COMP206_A2 directory. Pay very close attention to required file names **in bold**, because we require that you hand in files with precisely the same name to test your code automatically. Ensure you have a terminal (shell) with gcc installed.

How to Hand-in

Please make sure that you've run all of the tests mentioned in this document, plus some more that you can think of for each program.

As always, test on mimi.cs.mcgill.ca or the Trottier lab machines (they are identical, so either is fine). This is especially important for A2 as it's your first time trying the gcc compiler and so you need to discover any differences that might exist between your own computer's compiler, IDE etc and gcc on mimi. The official version for testing is gcc 5.5 (as exists on mimi and the Trottier labs).

Submit a single zip file to My Courses, through the Assignment 2 submission folder. Create this zip file with the command:
$ zip A2_solutions.zip **q1a_simple_diamond.c q1b_sierpinski_diamond.c q2_extract_wiki_links.c**

The submission deadline is 23:59 Tuesday, October 9th

Good luck!

Question 1: You are a diamond!

AASCI art is a classical way to show off your newfound programming ability. It means using the terminal printing abilities of your program to do something useful (or just to have fun, which is more our focus here).

Part A) Simple Diamond – 20 marks

Create the C program **q1a_simple_diamond.c** to produce a program which takes 1 argument, the height **H** of the diamond. Prints a diamond which is made up of **H** rows of asterisk (a.k.a. star, *) characters and spaces following this specification:

- The diamond's top and bottom rows must both be a single asterisk (same line if **H**=1)
- Subsequent rows must grow or shrink by 2 asterisks, according to the overall shape
- A single row in the middle of the diamond must have exactly **H** asterisks (no spaces in this one)
- The first asterisk in the middle row must appear at the very beginning of the line
- The asterisks in all rows must be perfectly centered with respect to the middle row

Your program is never allowed to segmentation fault, no matter how it is run. To prevent this, input checking must be performed:

- The program needs exactly one argument after the program name, no more and no less. Otherwise print: "ERROR: Wrong number of arguments. One required."
- The height argument must be a positive integer and odd (ensures the overall shape works). Otherwise print: "ERROR: Bad argument. Height must be positive odd integer."

```
dmeger@capilano:~/COMP206_A2$ gcc q1a_simple_diamond.c
dmeger@capilano:~/COMP206_A2$ ./a.out 5
  *
 ***
*****
 ***
  *
dmeger@capilano:~/COMP206_A2$ ./a.out
ERROR: Wrong number of arguments. One required.
dmeger@capilano:~/COMP206_A2$ ./a.out 2
ERROR: Bad argument. Height must be positive odd integer.
dmeger@capilano:~/COMP206_A2$ ./a.out 21
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
 *******************
*********************
 *******************
  *****************
   ***************
    *************
     ***********
      *********
       *******
        *****
         ***
          *
```

Part B) Sierpiński Diamond – 40 marks

A fractal refers to a recursively defined shape, such that infinite variations are possible at increasingly smaller scales. One such is known as the Sierpiński Triangle, and we will make our next diamond from two of these vertically mirrored. More information at: Wikipedia.

Create a second C program, **q1b_sierpinski_diamond.c**, which prints a modified diamond, such that each of the top and bottom half are Sierpiński Triangles. Your program should take two arguments: the height of the diamond, **H**, and the fractal level, **L**. Again, the diamond has a precise format:

- The level **L**=1 diamond must be identical to the simple diamond with the given height, **H**.
- For each level beyond 1, you must sub-divide top and bottom triangles using Sierpiński's rule. Replace a **H/2** triangle whose tip touches the bottom of the original with spaces. This leaves 3 triangles, each with height **H/2** and level **L**-1, one above the missing triangle, and two beside it.
- We must continue recursively sub-dividing each remaining triangle until we reach level **L=1.**

Your program is never allowed to segmentation fault, no matter how it is run. To prevent this, input checking must be performed:

- Height **H** must meet all the same conditions as above, with the same error messages.
- Height **H** must allow us to perfectly divide the triangle each time. This means that tri_height=ceil(**H**/2) must be a power of 2, with tri_height >= 2^{L-1}. Otherwise print "ERROR: Height does not allow evenly dividing requested number of levels."

```
$ ./a.out 7 2
   *
  ***
 *   *
*** ***
 *   *
  ***
   *
$ ./a.out 7 3
   *
  * *
 *   *
* * * *
 *   *
  * *
   *
$ ./a.out 7 4
ERROR: Height does not allow evenly dividing requested number of levels.
$ ./a.out 31 4
               *
              ***
             *   *
            *** ***
           *       *
          ***     ***
         *   *   *   *
        *** *** *** ***
       *               *
      ***             ***
     *   *           *   *
    *** ***         *** ***
   *       *       *       *
  ***     ***     ***     ***
 *   *   *   *   *   *   *   *
*** *** *** *** *** *** *** ***
 *   *   *   *   *   *   *   *
  ***     ***     ***     ***
   *       *       *       *
    *** ***         *** ***
     *   *           *   *
      ***             ***
       *               *
        *** *** *** ***
         *   *   *   *
          ***     ***
           *       *
            *** ***
             *   *
              ***
               *
$
```

```
$ ./a.out 15 4
       *
      * *
     *   *
    * * * *
   *       *
  * *     * *
 *   *   *   *
* * * * * * * *
 *   *   *   *
  * *     * *
   *       *
    * * * *
     *   *
      * *
       *
$ ./a.out 15 2
       *
      ***
     *****
    *******
   *       *
  ***     ***
 *****   *****
******* *******
 *****   *****
  ***     ***
   *       *
    *******
     *****
      ***
       *
$ ./a.out 15 3
       *
      ***
     *   *
    *** ***
   *       *
  ***     ***
 *   *   *   *
*** *** *** ***
 *   *   *   *
  ***     ***
   *       *
    *** ***
     *   *
      ***
       *
$
```

Question 2 Wiki Browsing – 40 marks

Wikipedia is a community developed encyclopedia that contains loads of useful information, if you know how to navigate it properly. You must create a C program, called **q2_extract_wiki_links.c**, that parses pages from the site Wikipedia using C's text processing functions found in <string.h>. Print the names of all links to other Wikipedia pages that you find to the terminal. Those links appear in this form:

*****easy to read description*****

You must match the bold text, but elements in italics can be anything. Please note that PageName is just an example. The real name will be something meaningful like Chicken. All of the following are valid:

- Chicken
- Chickens
- poultry

However, the following should not be matched (because something in the bold part is incorrect):

- Chicken (this is a link to a normal website off Wikipedia)
- Chicken (missing the title= section)

If your program works, every **PageName** printed should mean https://en.wikipedia.org/wiki/PageName is a valid web page. You can follow several of them to test, and see where your searches get you, e.g.,

```
$ gcc q2_extract_wiki_links.c
$ wget https://en.wikipedia.org/wiki/List_of_animal_names
$ ./a.out List_of_animal_names
Main_Page
Special:MobileLanguages/List_of_animal_names
Young_Animal_(DC_Comics)
Animal_epithet
File:Mother_sea_otter_with_sleeping_pup.jpeg
Sea_otter
Morro_Bay
Animal
Male
Female
Book_of_St._Albans
Juliana_Berners
Taxa
Family_(biology)
Class_(biology)
Clade
Female
Male
Collective_noun
Collateral_adjective
Binomial_nomenclature#History
Aves
Chicken
Bird
Bovinae
Herd
(output truncated)
```

```
$ wget https://en.wikipedia.org/wiki/Bird
$ ./a.out Bird
Wikipedia:Featured_articles
Wikipedia:Protection_policy#semi
File:Bird_(Intro).ogg
Bird_(disambiguation)
Birds_(disambiguation)
Aves_(disambiguation)
Avifauna_(disambiguation)
Early_Cretaceous
Holocene
Precambrian
Cambrian
Ordovician
Silurian
Devonian
Carboniferous
Permian
Triassic
Jurassic
Cretaceous
(output truncated)
```

```
$ wget https://en.wikipedia.org/wiki/Jurassic
$ ./a.out Jurassic_Park
File:Jurassic_spoken_article.ogg
Precambrian
Cambrian
Ordovician
Silurian
Devonian
Carboniferous
Permian
Triassic
Cretaceous
Paleogene
Neogene
Oxygen
Carbon_dioxide
Parts_per_million
Template:Jurassic_graphical_timeline
Mesozoic
Triassic
Cretaceous
Early_Jurassic
Middle_Jurassic
Late_Jurassic
Hettangian
Sinemurian
(output truncated)
Jurassic_Park
(output truncated)
```