Week 09, Homework 07	Weight: 4%	Due: Monday week 10, 09:00am (via sync)

Pre-homework Preparation:

- **Lab: Week 08**
- **Lectures: Weeks 01, 02, 03, 04, 05, 06, 07, Monday/Tuesday 08**

Homework Activities:

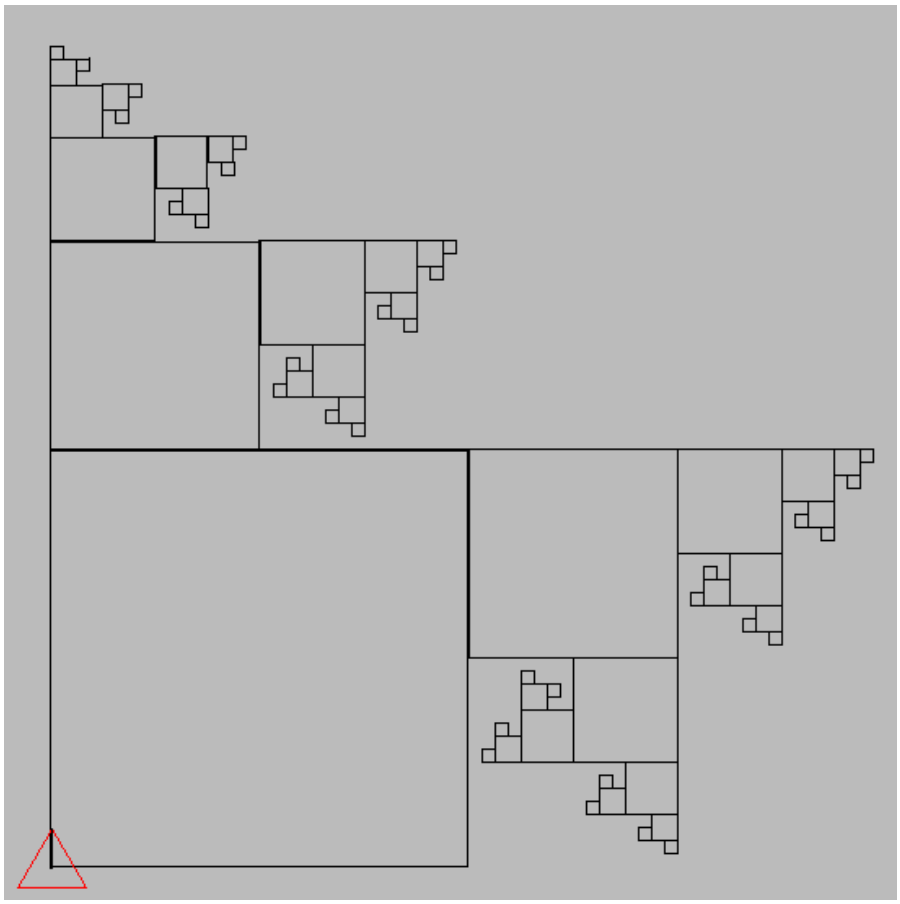
Obtain the homework files using **sync**. For each question's source file, be sure to fill in the file comment header accurately!

Question 1: Box fractal

A fractal is a shape that contains a repeating shape at different scales. Below is a screenshot of a simple fractal.

Fractals are good candidates for recursive algorithms, because the problem of drawing them overall has as a subproblem drawing them at a smaller scale. Of course for such algorithms to terminate there must be a smallest size below which we stop drawing the fractal - the size of the smallest box in the image.

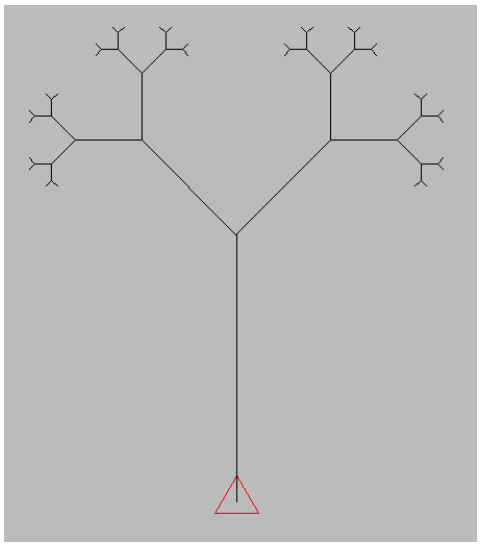
Add code to **`hw07q01.c`** to draw the above figure using a *recursive* algorithm. The boxes decrease in size by half each time, and the smallest boxes have sides of length 8.



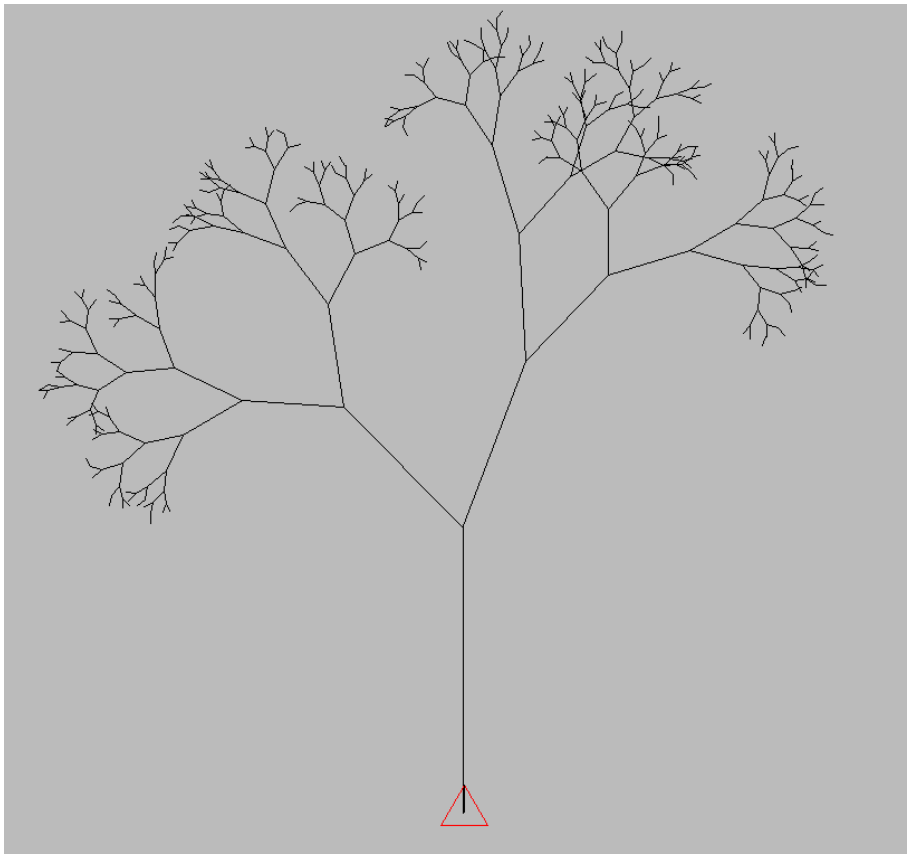
Question 2: Invent your own fractal

Following the general pattern of the algorithm that you implemented for the previous question, implement your own fractal.

You can make your fractal more interesting by varying shape, colour and scale as you recurse, and by introducing some degree of randomness. For example, some trees and plants have a fractal shape consisting of a stalk with two branches.



By varying the angle and length and number of these somewhat at random the image can be made to look more natural.



Question 3: Returning a dynamically allocated string

The code in **hw07q03.c** does not work as intended. The function **get_user_input()** is supposed to make use of **scanf()** to return a string that the user has entered at the console.

```
char input_buffer[1000];

char* get_user_input()
{
    scanf(" %s", input_buffer);
    return input_buffer;
}
```

scanf() needs some memory in which to place the string that is read. A pre-allocated array of characters or bytes of a fixed size is conventionally called a *buffer*. This cannot be allocated locally to **get_user_input()** because any returned pointer to the buffer would be invalid after the function returned and its stack frame was released. So the implementation makes the buffer global so that it will persist after the function call. However, even this causes problems as you will discover by compiling and running the program and experimenting with various inputs.

Modify the code so that instead of returning a pointer to the input buffer, the **get_user_input()** function first makes a copy of the string into a dynamically allocated character array of the appropriate size, and then returns a pointer to this memory instead. You can use the standard library function **strlen()** to help you, and of course **malloc()**.

Question 4: Printing a substring

In **`hw07q04.c`** complete the implementation of the function **`get_substring()`** that takes two pointers to chars, assumed to reference elements within the same character array (string). The function should return a new null-terminated string starting with the character pointed to by the first pointer, and all subsequent characters up to, but not including the second character.

Tests are given and can be run using **`make test`**. For full marks you should also define your own **`main()`** function that calls the **`get_substring()`** function to implement dialogs such as the following.

```
Please enter a string: rhinoceros
Index of first character of substring?: 0
Length of substring?: 5
Substring is "rhino"
```

Question 5: Making space

In **`hw07q05.c`** implement a function called **`insert_space()`** that takes as arguments a string and a location within that string, and a number of spaces to insert at that point, and returns a pointer to a copy of the original string with the right number of spaces inserted at the right location. For example…

```
insert_space("helloworld", 5, 3);
```

…should return…

```
hello   world
```

Tests are given and can be run using **`make test`**. You can also define your own **`main()`** function for your own testing purposes.

Question 6: Fix the Memory Leaks

The program in **`hw07q06.c`** has memory leaks that will cause it to crash after a time. Find them, and fix them!

A good experiment to do is to build and run the program, then while it is running log in on a different virtual terminal (use CTRL-ALT-RIGHT_ARROW to switch, and CTRL-ALT-LEFT_ARROW to switch back) and run the command **`top`**. This will show you all processes running, and you will be able to watch the program gradually consuming more and more memory, until it runs out!

Note that this program uses a somewhat silly algorithm to generate its output - in fact it is not necessary to precompute the results in memory and then print them. However, this program is simulating a situation in which blocks of memory *do* need to be used, for example when loading and processing the contents of multiple files.

Therefore in fixing the code you should retain this behaviour. In particular retain the signatures of each function defined in **`hw07q06.c`** and the overall order of function calls - not including calls to standard library functions - while fixing memory leaks and possibly eliminating unnecessary behaviour within functions.

Remember that the **`free()`** function is used to release dynamically allocated memory, but should only be used on a pointer once the memory pointed to is no longer needed.

Question 7: Simple text editor

We can think of a simple text document, such as a C source code file, as being an array of strings, one string for each line. This could be implemented using an array of pointers to chars.

A simple *line-oriented* text editor could then implement a command loop whereby the user enters one of the following commands

- a - append a line to the end of the current document
- i - insert a line before a specified line (giving the line number)
- d - delete a line
- l - show the current document with lines numbered
- e - edit a numbered line, followed by:
    - d - delete characters in a given range
    - i - insert characters in a particular position
    - a - append characters
- q - quit

Implement such a program in **`hw07q07.c`**. Below is a possible sample session.

```
> a
text? Hello
> l
1. Hello
> a
text? World
> l
1. Hello
2. World
> i
line? 1
text? I'd like to say
> l
1. I'd like to say
2. Hello
3. World
> e
line? 2
    01234
2. Hello
command? i
position? 3
text? lll
> l
1. I'd like to say
2. Hellllo
3. World
> q
Bye!
```

Homework Submission:

Run the **sync** command to submit your completed homework.

Marking Criteria:

Have you completed each of the following?

Marking Criteria:	Week 07, Homework 06 Weight 3%	Maximum Possible Mark:	Mark achieved?
Q1:	Box fractal drawn	2	
	Recursive algorithm used	4	
	Correct dimensions for fractal	2	
	Code well formatted, named and commented	2	
Q2:	Really cool fractal drawn	4	
	Recursive algorithm used	4	
	Code well formatted, named and commented	2	
Q3:	Code refactored so that input strings are output correctly	4	
	Dynamic memory allocation used	4	
	Code well formatted, named and commented	2	
Q4:	All tests pass	4	
	Own main() written with calls to `get_substring()`	4	
	Code well formatted, named and commented	2	
Q5:	All tests pass	6	
	Own main() written with calls to `insert_space()`	2	
	Code well formatted, named and commented	2	
Q6:	Memory leaks fixed preserving overall behaviour of the program	4	
	Code well formatted, named and commented	2	
Q7:	Editor works as described	6	
	Code well structured and formatted	2	
Total:		**70**	