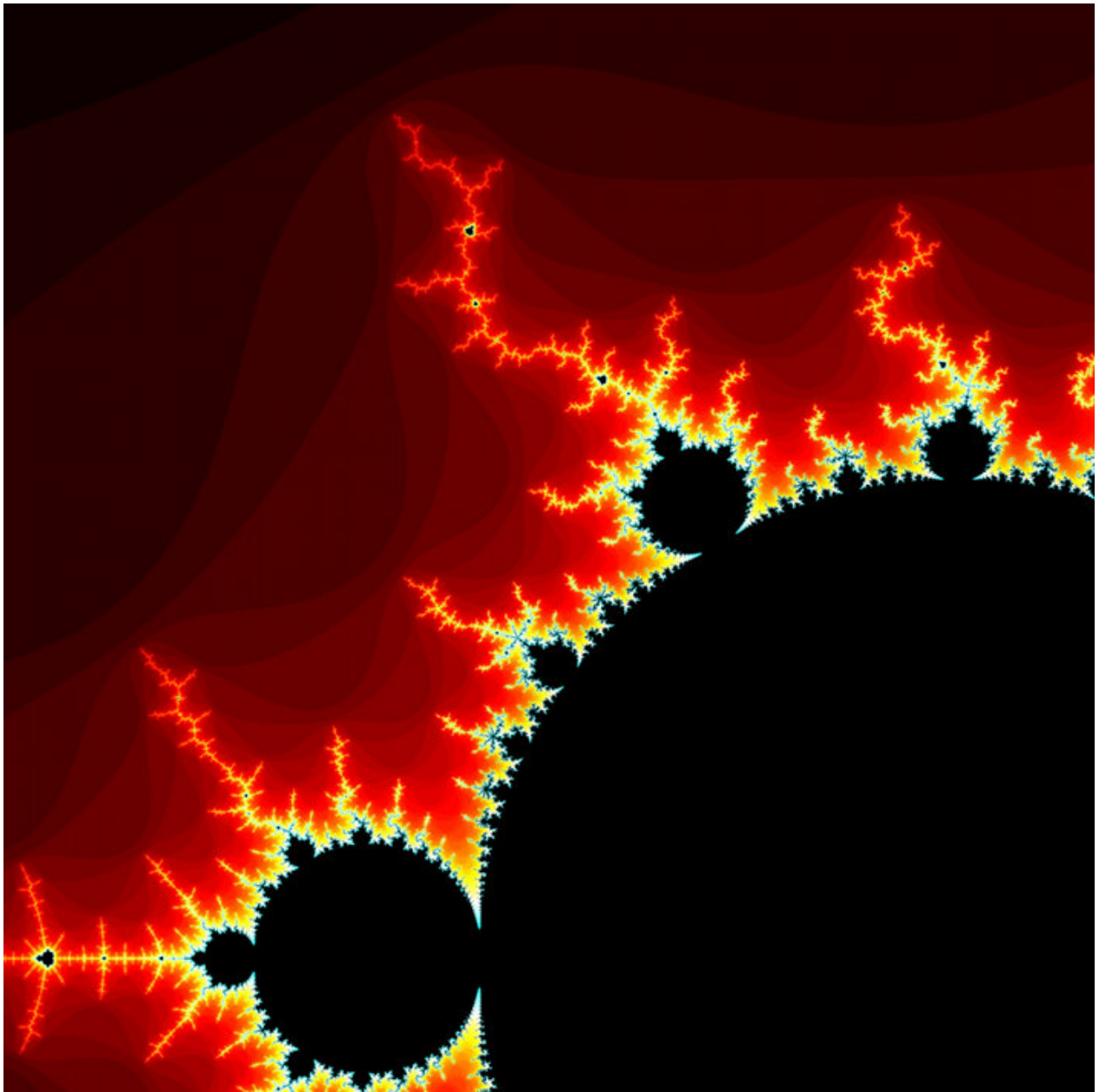


Assignment 1: Mandelbrot

This is a team assignment,
to be done with your week 5-8 lab partner.

This assignment is due on
Sunday 17th September,
at **11:59:59pm** (Sydney time).

The Mandelbrot set, named after its inventor [Benoit B. Mandelbrot](#) is astonishingly complex and detailed at all levels of abstraction.



It has breathtaking beauty. It always reminds me of the glorious complexity and ultimate unknowability of the universe. It will probably always remind you of Assignment 1 in your first computing course.

Objectives

- You produce a working correct Mandelbrot image server which you can use to explore the Mandelbrot set.
- You work effectively in a pair.
- You successfully manage this multi-part project.
- You write clear, high-quality, correct code.
- You use your code to produce your own striking and impressive images of the Mandelbrot set and share these with others.
 - To do this you will need to explore and discover/select interesting regions of the set.
 - Then you will need to design an effective color map to display the features of the region in the most effective way you can, to produce the most beautiful images you can.

Summary

Step 1: You are to work in with your lab partner to write a web server which will serve images of small regions of the Mandelbrot set (given a location and the level of zoom to use).

Step 2: You will then use this tool to explore the Mandelbrot set and produce images of your most interesting journeys and most beautiful discoveries.

This task is *similar* to Assignment 0 in that we expect you to:

- produce **correct, clearly-written**, and **rigorously tested** C code, and
- for this to be **delivered on time**.

This task is *different* to Assignment 0 in that

- it is to be **done in a team**;
- we expect you to keep a **detailed blog** of your progress with respect to the **server's design and programming** and **teamwork and project management**; and
- we are not only interested in the software you write, but also **how you use it to solve a problem**: to **explore** and **document** interesting parts of the Mandelbrot set.

We will be assessing five aspects of your assignment:

- the **correctness** of your code;
- your **programming style**;
- your **initial plan** for the assignment,
- your regular **progress blogs**; and
- your **final reflection** at the end.

Before you begin...

Working with your partner

Exchange contact details with your lab partner, and arrange how you will meet and work together over the next two weeks.

Making a plan

Once you've worked out how you'll communicate, you should sketch out, with your partner, how you plan to approach the task.

This should take the form of a blog post on WebCMS 3, and should include how you're going to approach the task, how you've planned to divide work between yourselves, who will be responsible for what portion of the code, and a rough timeline of how you'll tackle the problem.

There is a blog template that you should use to write your plan. There are general instructions available [here about how to create a blog post from a template](#).

You might also like to consider the discussion you had in your tutorial this week about assignment 0, and what you learned from it that you plan to bring into the next assignment.

Understanding the problem

As you no doubt found in Assignment 0, it's important to make sure you've fully understood the problem before you try to start writing a solution.

Get the files

You are provided with two header files for this assignment:

- [mandelbrot.h](#)

defines constants `TILE_SIZE` , `MAX_STEPS` , and `NO_ESCAPE`

defines structures `pixel` and `complex`

declares prototypes for `drawMandelbrot` , `escapeSteps` , `escapeGrid`

- [pixelColor.h](#)

defines prototype for `pixelColor`

You are also provided with several stub files for this assignment:

- [mandelbrot.c](#)

This should contain all of your Mandelbrot-related code.

- [pixelColor.c](#)

This should contain all of your color-related functions.

- [server.c](#)

This should contain all of your server-related code, including the `main` function for your server.

All of these `.c` files should `#include "mandelbrot.h"` .

In addition to this, `mandelbrot.c` and `pixelColor.c` should also `#include "pixelColor.h"` .

Implement the function prototypes from the `.h` files in the correspondingly named `.c` files. Make any additional helper functions you write `static` . Don't change the `.h` files: we'll use our own copy of the `.h` files when marking – you'll only submit the three `.c` files.

Progress Blogs

Like all good scientists you should keep a record of your adventures, trials, tribulations, and accomplishments.

For this assignment, you should make regular, **individual** blog posts on the two parts of this assignment: the *scientific and technical* portion, and the *teamwork and project management* portion.

You should have a roughly even split between these two themes. Your blog posts don't have to be novels, but should try to communicate the particular topics that are relevant to each theme.

Blogs on the **scientific and technical** portion should discuss the progress you make in developing your solution, technical aspects of your server's design, the bugs you've encountered, how you fixed them, and discoveries you've made in the Mandelbrot set.

Blogs on the **teamwork and project management** portion should discuss how well your plan is going, and what changes you've made to your plan. This may include how your division of work went,

More information on progress blogs is coming soon.

Completing the assignment

Step 1: The Server

Make sure to also check out the [how to approach this task](#) page! The content below is just a high-level overview of the task.

Write an HTTP server to serve image tiles showing a part of the Mandelbrot set. Base your server on the supplied `server.c` below. This is based on the poetry server from Week 6 labs.

Your server is to serve up 512 x 512 pixel BMP images. Given the x, y co-ordinates (doubles) and a zoom level (an int) the server returns a 512x512 pixel tile of the region of the Mandelbrot set centered at that point.

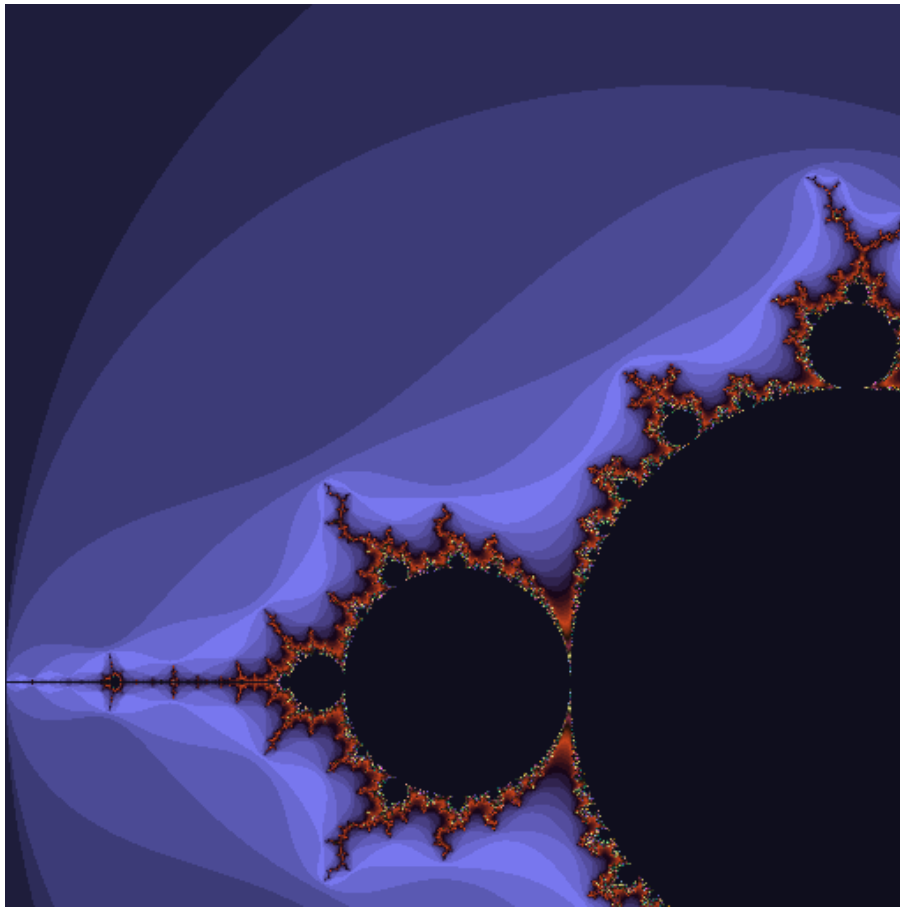
The desired `x`, `y`, and `zoom` will be encoded in the URL of the tile file requested from the server. The URL is structured:

```
http://localhost:1511/mandelbrot/2/zoom/x-coord/y-coord/tile.bmp
```

For example, if you typed:

```
http://localhost:1511/mandelbrot/2/8/-1.0/0.5/tile.bmp
```

in your web browser, then the server should return an HTTP response containing an image of the region of the Mandelbrot set with center (-1.0, 0.5) and zoom level 8 (see below for how to interpret the zoom level). That particular tile should look something like this:



And if you typed

`http://localhost:1511/`

without any file name at all, your server should just respond with the HTML code:

```
<script src="http://almondbread.cse.unsw.edu.au/tiles.js"></script>
```

This code will cause your browser to download and display a funky viewer (like [this!](#)) with which to navigate your beautiful Mandelbrot set.

You'll need to return a `text/html` HTTP response header first, just like your poetry server, then a blank line, then the HTML code above. Here is some [sample code](#), showing how my server does it.

IMPORTANT NOTE:

The viewer code will not work unless your code can successfully serve images by browsing directly to them. So do this step only after you've gotten the previous step working!

The Region

The x , y co-ordinates give the center of the image. The zoom level z gives the distance between pixels as follows:

$$\text{distance between pixels} = 2^{-z}$$

So, given $zoom = 8$, $x = -1.0$, $y = 0.5$, the distance between adjacent pixels in the image returned will be $2^{-8} = 1/256 = 0.00390625$, and the requested image is for a region of the set 1.9960938 wide by 1.9960938 high (as $511 * 0.00390625 = 1.9960938$), centered at $(-1.0, 0.5)$.

Pixel iterations, without complex numbers

To generate an image of part of the Mandelbrot set follow the process we discussed in lectures for each pixel in the image: find the (x,y) co-ordinates of the pixel's position in the region, and then iterate the Mandelbrot equation:

$$\text{Point}_n = (\text{Point}_{n-1})^2 + (x,y)$$

starting with $\text{Point}_0 = (0,0)$ until you generate a point Point_n which is 2 or more away from the origin, or until you have performed 256 iterations.

Pixel iterations, with complex numbers

If you'd prefer a more complex solution, try this: pretend the region is an Argand diagram, a diagram of the complex plane.

As before, when we find the (x,y) co-ordinates of the pixel's position in the region, we've actually found a complex number, $\mu = x + y \mathbf{i}$. Now, for any integer $n \geq 1$,

$$\begin{aligned} z_0 &= 0 + 0 \mathbf{i} \\ z_n &= z_{n-1}^2 + \mu \end{aligned}$$

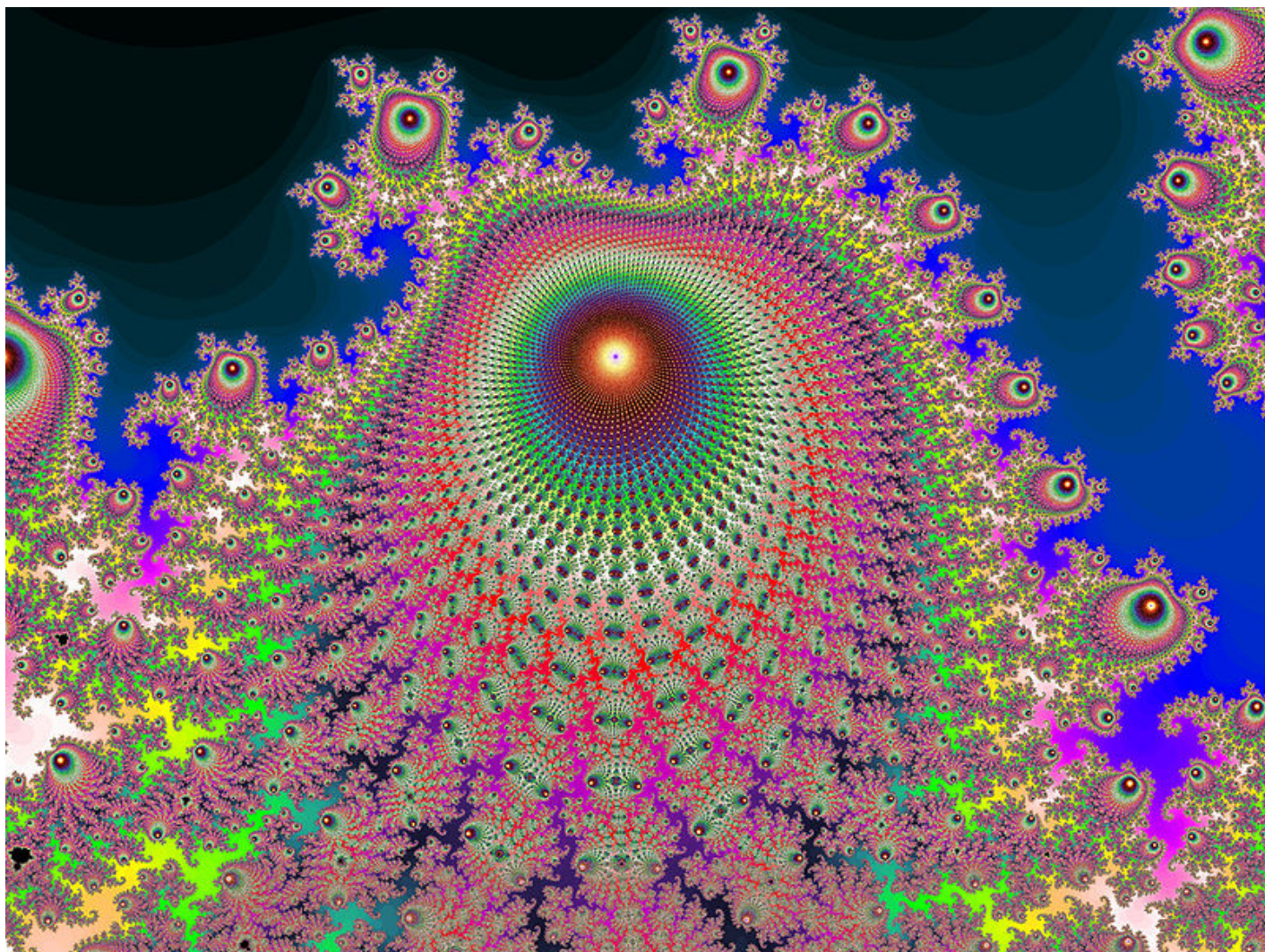
If the magnitude of this new value, $|z_n| > 2$, the point has escaped. Rinse and repeat.

`escapeSteps` should then return the last n before the value escaped, or 256, whichever is smaller.

You might find these facts useful, for any complex number z :

$$\begin{aligned} \mathbf{Re}(z^2) &= \mathbf{Re}(z)^2 - \mathbf{Im}(z)^2 \\ \mathbf{Im}(z^2) &= 2 \mathbf{Re}(z) \mathbf{Im}(z) \\ |z| &= \sqrt{(\mathbf{Re}(z))^2 + (\mathbf{Im}(z))^2} \\ |z|^2 &= \mathbf{Re}(z)^2 + \mathbf{Im}(z)^2 \end{aligned}$$

Step 2: Discover... and Color!



Source: [Wikipedia](#), original upload 24 November 2005 by David R. Ingham CC-BY-SA 3.0

You may choose how to colour the pixels yourself, using your artistic, analytical and visual design skills. Some suggestions are given below, there are many ideas on the internet, and we'll discuss this more in lectures also.

The basic idea is to colour each pixel based on the number of iterations the above process takes.

The simplest approach is to colour a pixel black if you took 256 iterations, white otherwise. This will give you a black and white image.

A more interesting way of colouring the pixels is to shade them grey – with intensity depending on the number of iterations.

The most powerful and effective way of colouring the pixels is to use a different RGB colour for each possible number of iterations. That is how the images on this page were generated. The artistry here lies in picking the colours and deciding how they correspond to the different iteration counts.

Put your color decision code into `pixelColor.c` . Put the coloring-in logic into the `pixelColor` function we discussed in lectures (the prototype is in `pixelColor.h`).

If you put any other helper functions into `pixelColor.c` you must make them static. (put the word "static" at the start of the functions' declaration and definition).

Submit your best images (and the corresponding `pixelColor.c` files) to the MandelbrArt competition (Activity released in Week 8).

After the assignment is complete, in the Week 9 tute-lab, we will get each lab pair to show their best images to the rest of the class, and the tutorial will vote to determine which image will represent their class. Each tutorial will submit their best image, and the whole course will vote to determine the most amazing Mandelbrot Art created in COMP1511, 17s2. The winning lab pair(s) will receive a small prize to acknowledge their fantastic work :)

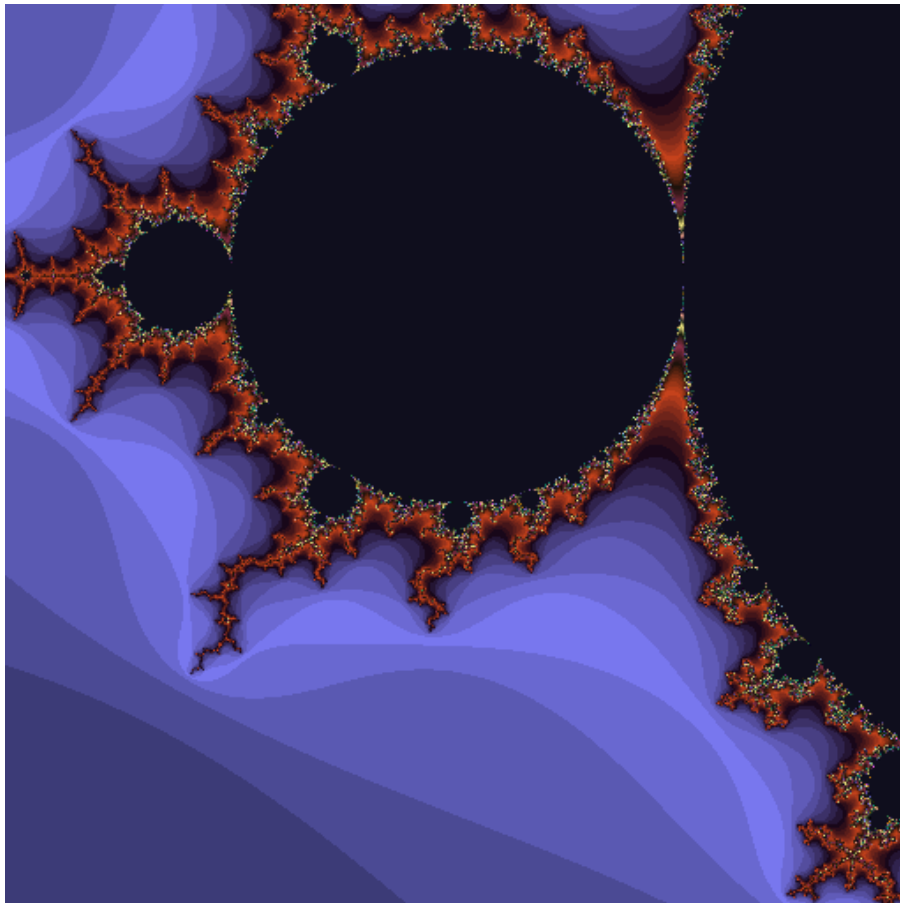
Testing / Debugging

You can use the browser linked from the lab pages to browse the set interactively. To just get single tiles and compare them with the tiles you generate you can ask the sample Mandelbrot server we have left running on the internet. Amusingly it's called the Almond Bread server – a little jokeule of Julian's (he wrote the first version for us)

For example: to inspect the tile centred on $(-1.0, -0.2)$ at zoom level 9, request the following image in your browser address bar:

<http://almondbread.cse.unsw.edu.au/mandelbrot/2/9/-1.0/-0.2/tile.bmp>

That url fetches this tile:



Submitting

You and your partner make a joint submission, as with lab tasks. As long as your lab pair group is set up correctly, only one of you needs to submit in order for both of you to be able to see your submission.

You should submit `mandelbrot.c`, `server.c`, and `pixelColor.c`. You don't need to upload any of the `.h` files, as these are already provided to the automarker.

Submit your work with the `give` command, like so:

```
$ give cs1511 assign1 mandelbrot.c pixelColor.c server.c
```

Or, if you are working from home, upload the relevant file(s) to the `assign1` activity on [Give Online](#).

The files `mandelbrot.c` and `pixelColor.c` should *#include* *and implement* the provided [pixelColor.h](#) and [mandelbrot.h](#) exactly.

The deadline for this task is Sunday of Week 8. Submit your three `.c` files on or before the deadline.

Assessment

This assignment will contribute 10% to your final mark.

Mark breakdown

- 30% correctness
- 30% style
- 10% planning
- 20% progress blogs
- 10% final reflection

Correctness

30% of the marks for this assignment will come from the correctness of your program, i.e. what proportion of the autotests your program passes. These autotests **will not** be available to you during the course of the assignment, and will only be run after the due date of the assignment. Therefore, it is in your best interests to write your own extensive unit tests, so that you can catch any bugs in your program before we run our tests against it.

Style

30% of the marks for this assignment will come from hand-marking of the readability of the code you have written. These marks will be awarded on the basis of clarity, commenting, elegance, and style. In other words, your tutor will assess how easy it is for a human to read and understand your program.

Planning

10% of the marks for this assignment will come from the initial plan you create. More details on what is required can be found in the earlier section on planning.

We hope that, by writing a solid plan before you commence your assignment, you will find it easier to get started on the assignment, and that you will have thought through the design of your program, and that you will have hopefully anticipated possible problems you might run into, which will make them easier to overcome when they do occur.

As such, we will be assessing that your plan:

- is specific
- is realistic
- is likely to be helpful
- takes into account your individual circumstances
- anticipates problems and identifies in advance ways to overcome them

Progress Blogs

20% of the marks for this assignment will come from your progress blogs you write throughout the project. More details on this can be found in the earlier section on progress blogs..

Final Reflection

10% of the marks for this assignment will come from a retrospective reflective blog post you write after the assignment is complete.

Note: the Final Reflection will be due at 23:59:59 on Wednesday 20th September (three days after the due date for the rest of the assignment).

Late Penalty

The assignment is due at 23:59:59 on Sunday 17th September, Australian Eastern Standard Time.

If your assignment is submitted after this date, then for each hour it is late, the maximum mark it can achieve is reduced by 1%. For example, if an assignment worth 74% was submitted 12 hours late (noon on Monday), the late submission would have no effect. If the same assignment was submitted 30 hours late (6am on Tuesday), the mark would be reduced to 70%, the maximum mark it can achieve at this time.

Plagiarism

Note: although this is a group assessment, plagiarism rules still apply.

Penalty	Description
-70%	Knowingly providing your work to anyone (outside of your group) and it is subsequently submitted (by anyone).
-70%	Submitting any other person's work (outside of your group). This includes joint work (across several groups).
0 FL for COMP1511	Paying another person to complete work. Submitting another person's work without their consent.

Resources

- [How to approach this task](#)
- Make sure you complete the poetry server and bmp image activities from previous weeks
- [pixelColor.h](#)
- [mandelbrot.h](#)
- Deliver the Mandelbrot viewer: [sample code](#)
- The second part of the task
 - [MandelbrArt](#)

changelog

v1.0.1: updated submission directions

v1.0.0: released to unsuspecting students!

To run Styl-o-matic:

```
$ 1511 stylomatic mandelbrot.c pixelColor.c server.c
Looks good!
```

You'll get advice if you need to make changes to your code.

Submit your work with the *give* command, like so:

```
$ give cs1511 assign1 mandelbrot.c pixelColor.c server.c
```

Or, if you are working from home, upload the relevant file(s) to the `assign1` activity on [Give Online](#).