COMP1511 17s2

— Lecture 15 —

Working with ADTs

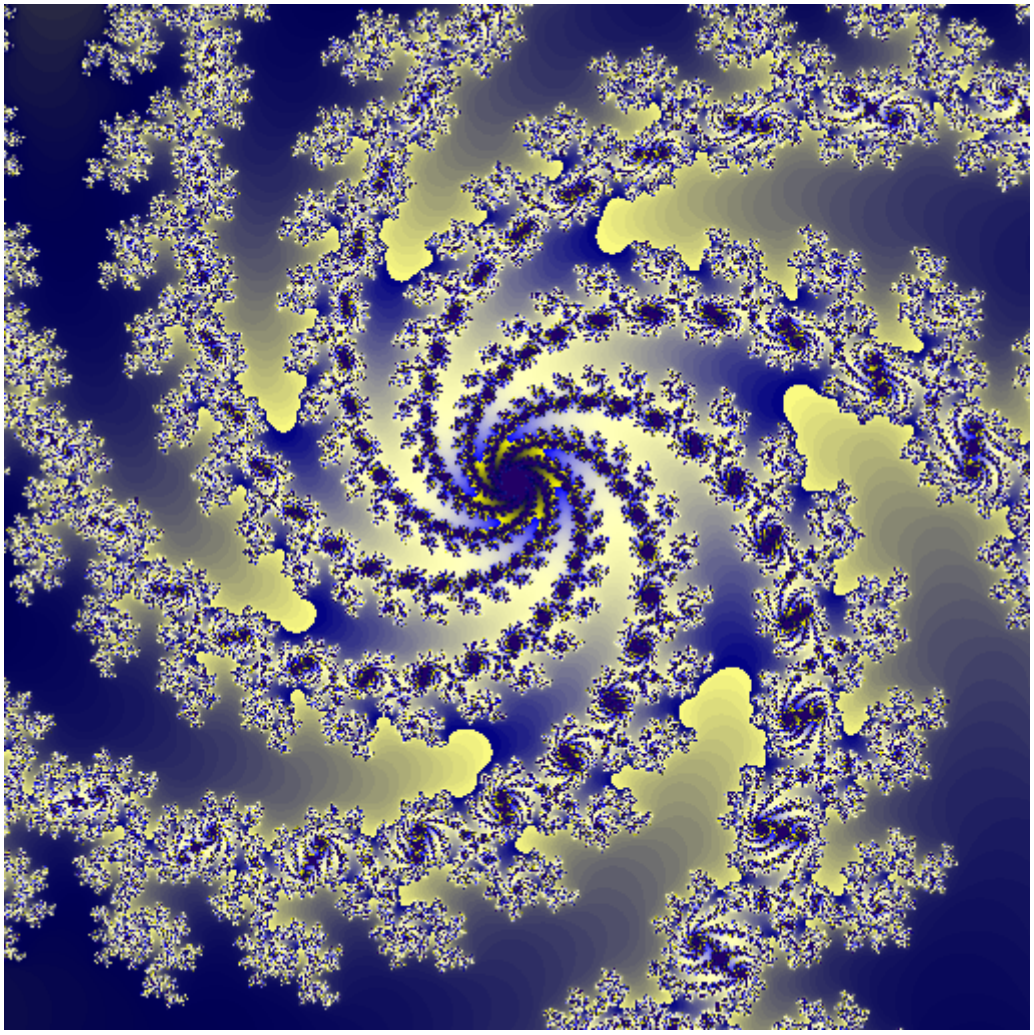
Andrew Bennett
<andrew.bennett@unsw.edu.au>

abstract data types
boundaries and testing

Don’t panic!

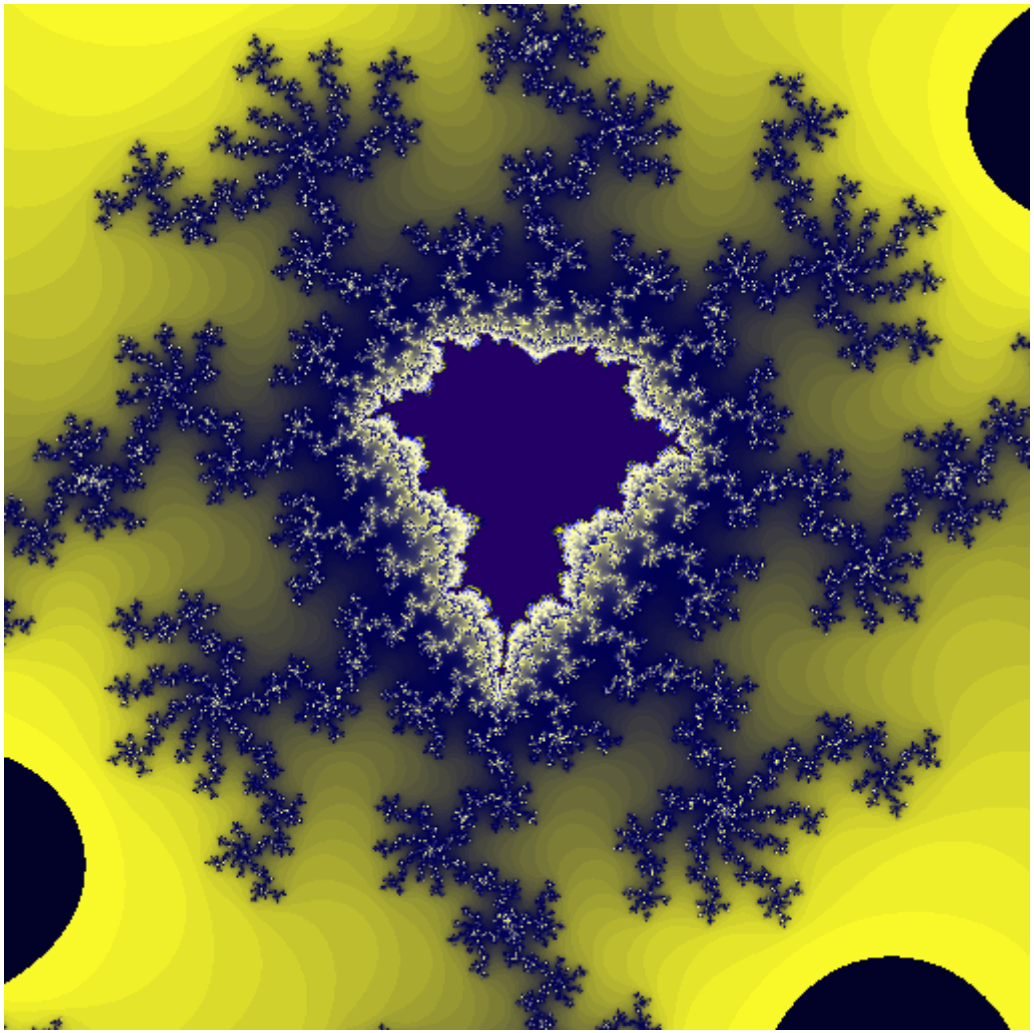
assignment 1
due Sun 17 Sep, 23:59:59 (Sunday, week 8)

searching for Xanadu…



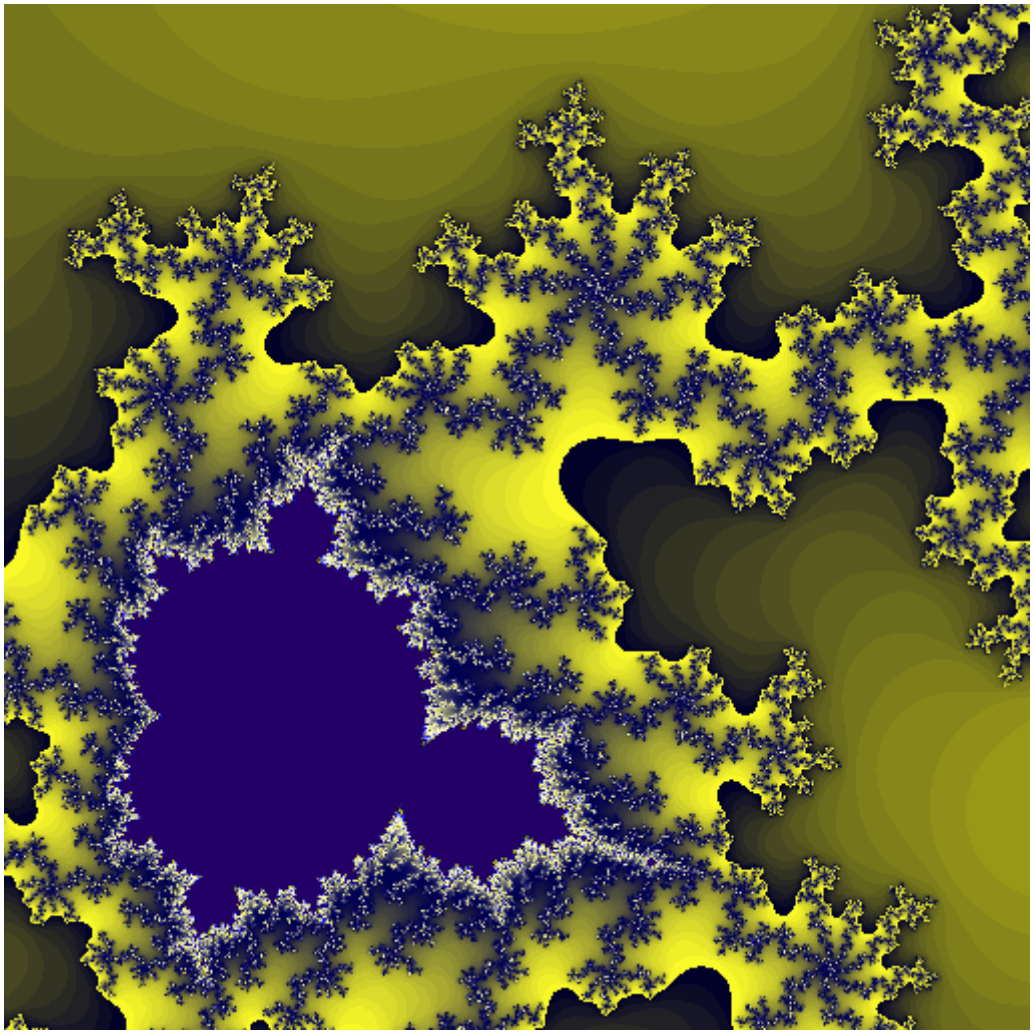
… mandelbrArt open now!

searching for Xanadu…



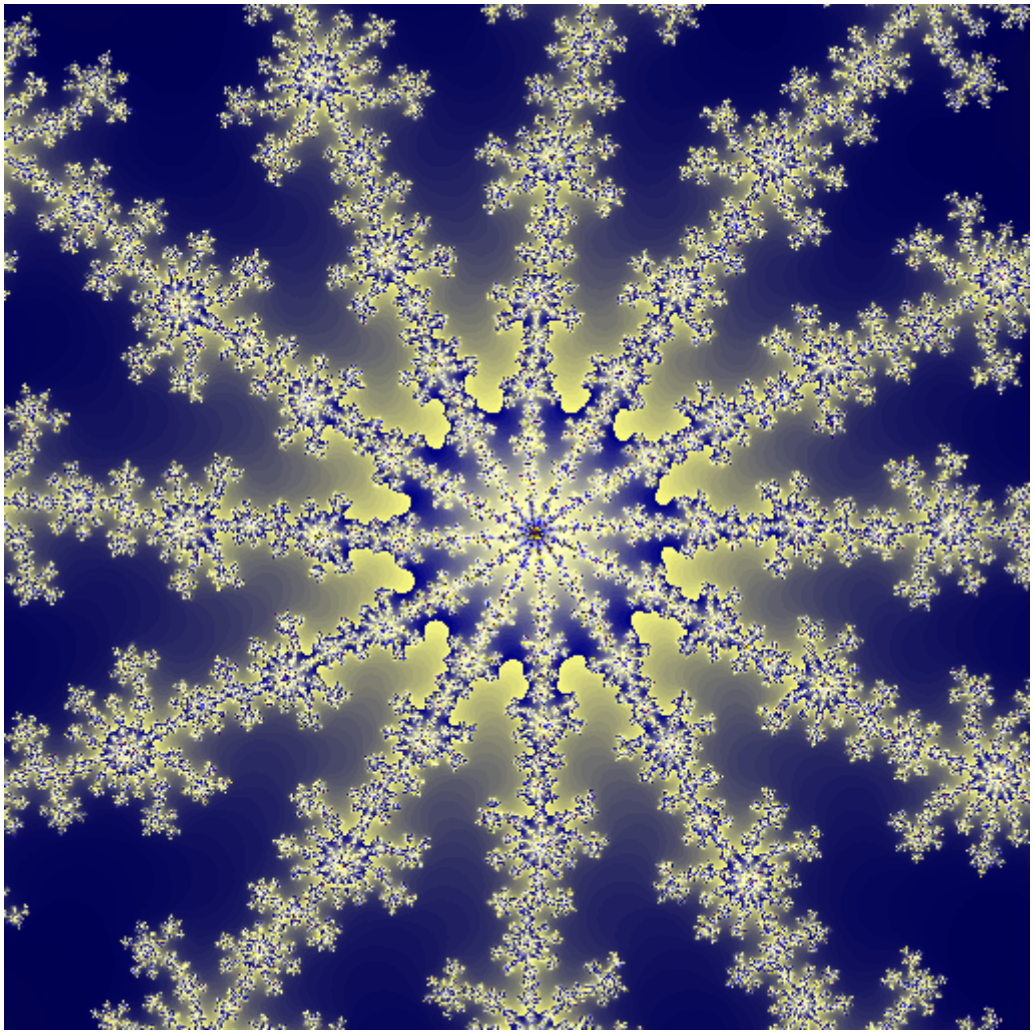
… mandelbrArt open now!

searching for Xanadu…



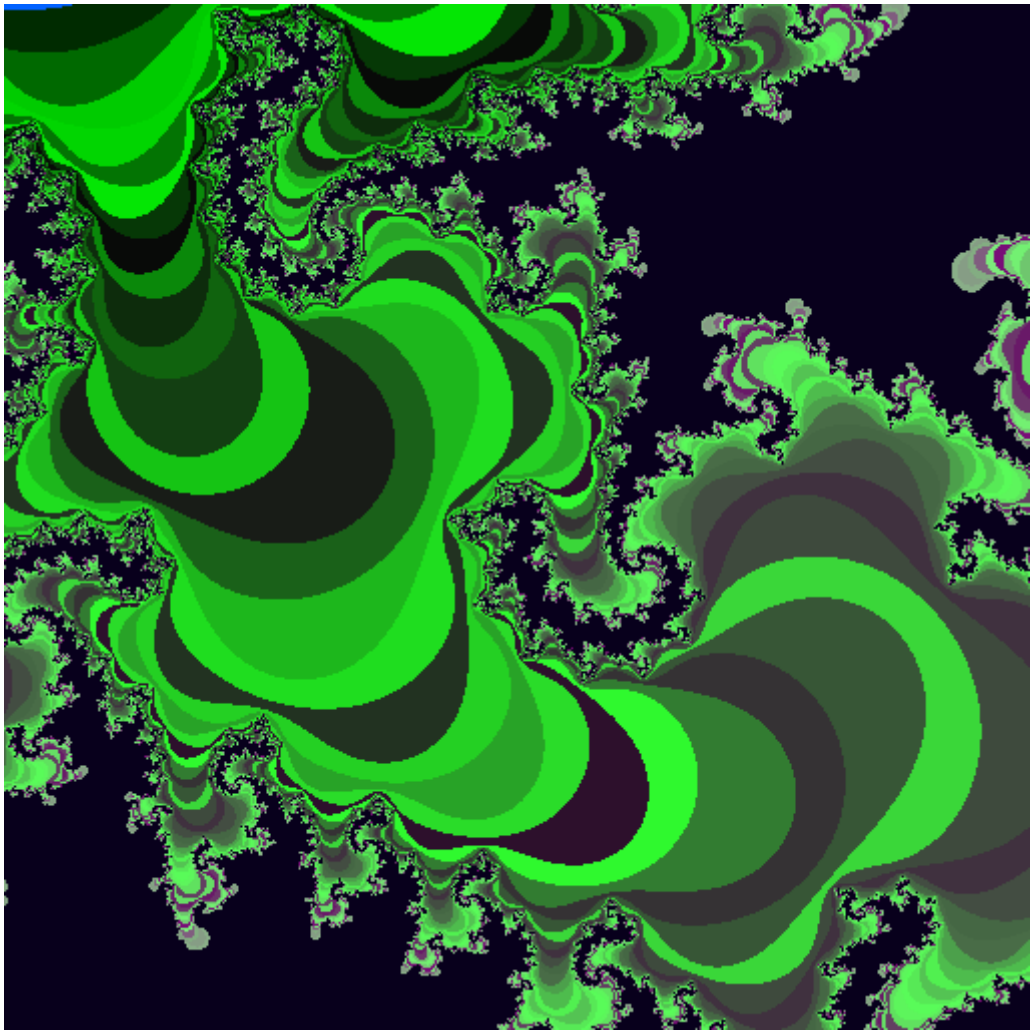
… mandelbrArt open now!

searching for Xanadu…



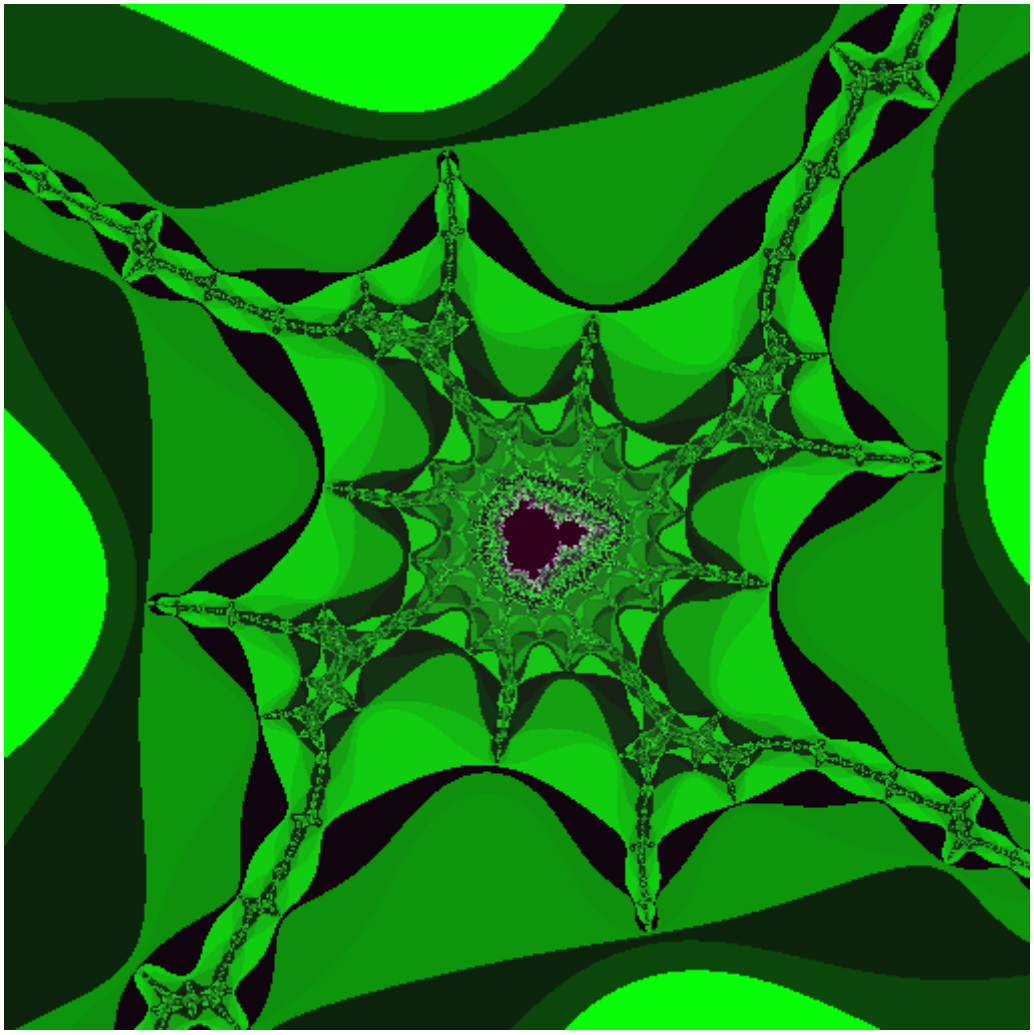
… mandelbrArt open now!

searching for Xanadu…



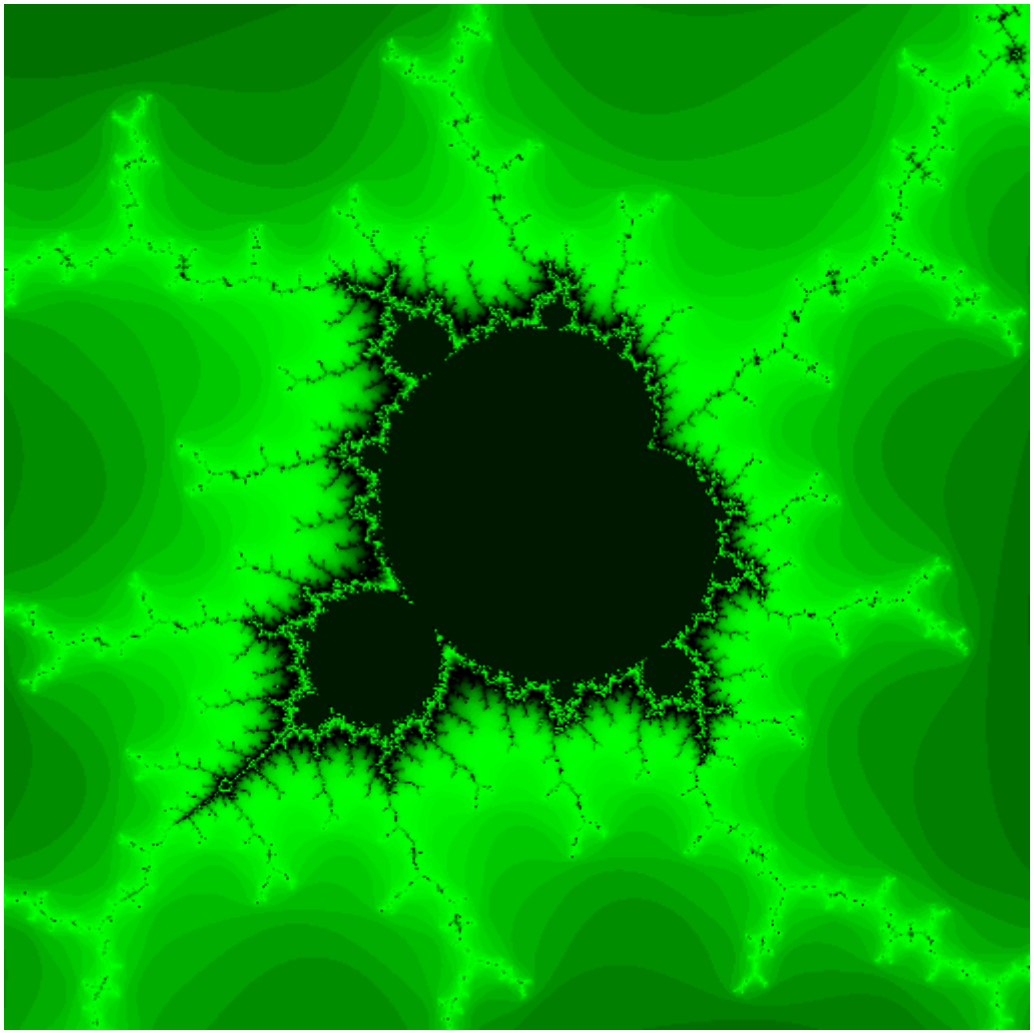
… mandelbrArt open now!

searching for Xanadu…



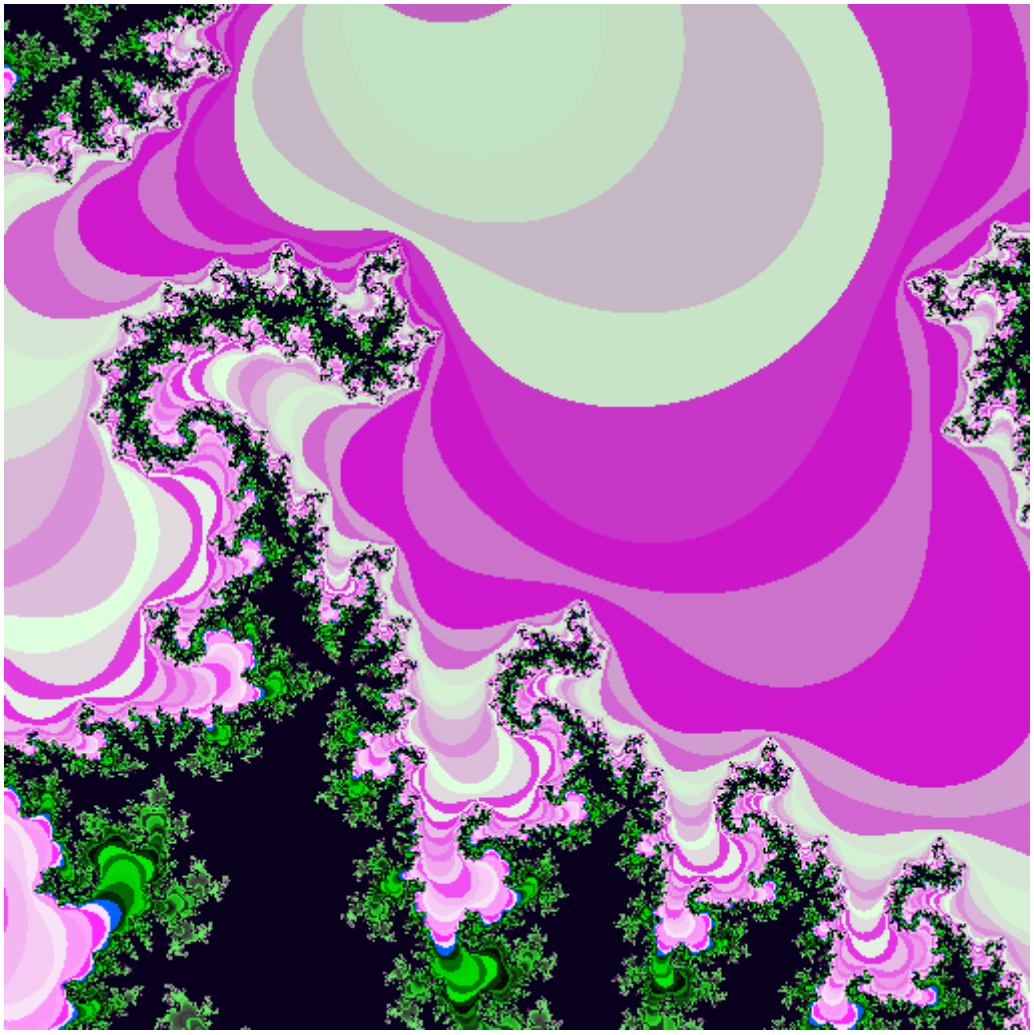
… mandelbrArt open now!

searching for Xanadu…



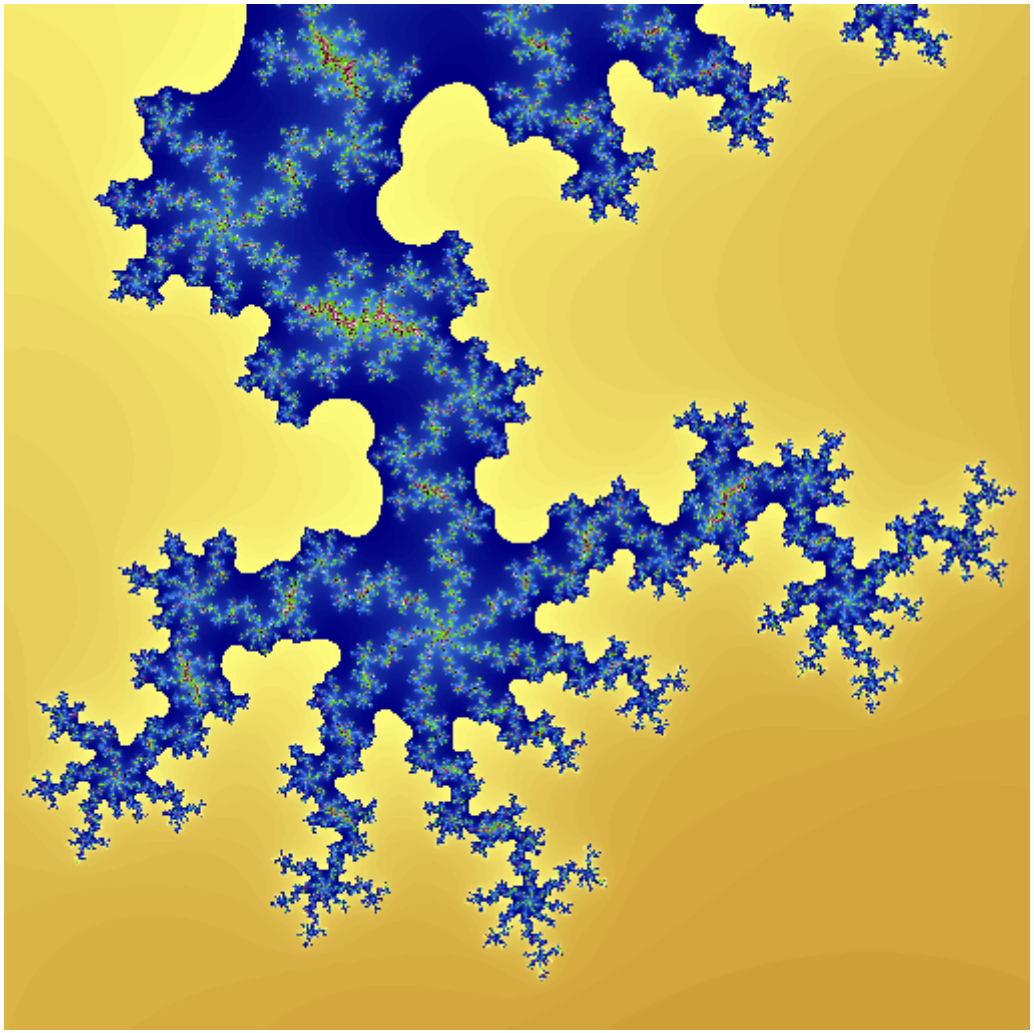
… mandelbrArt open now!

searching for Xanadu…



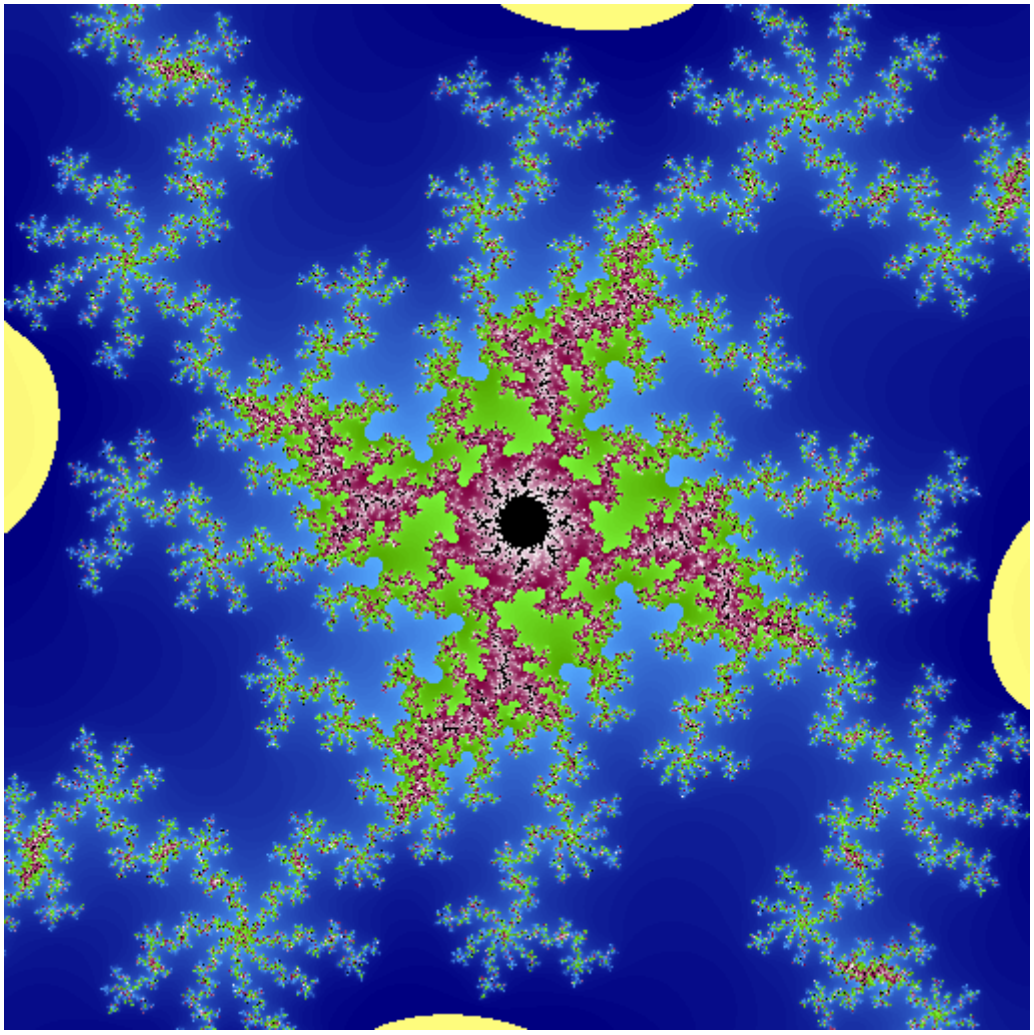
… mandelbrArt open now!

searching for Xanadu…



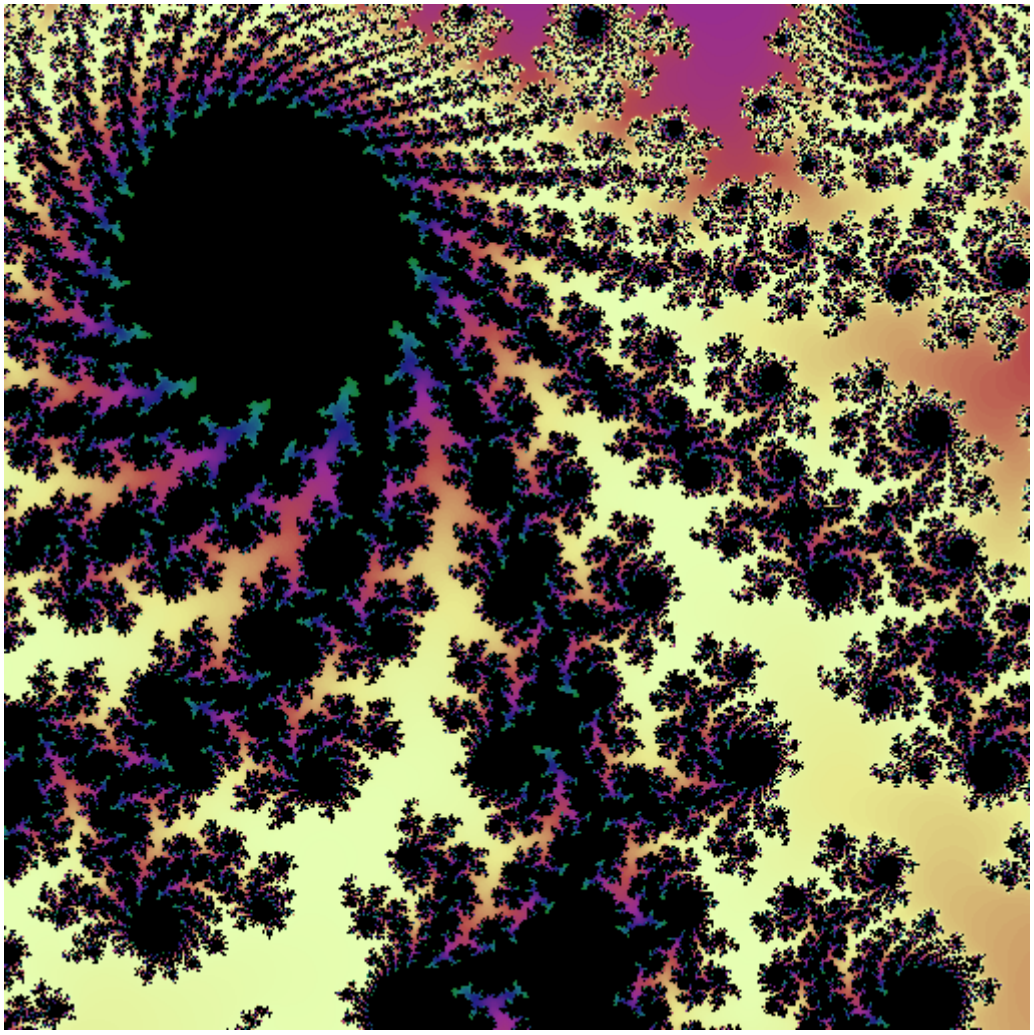
… mandelbrArt open now!

searching for Xanadu…



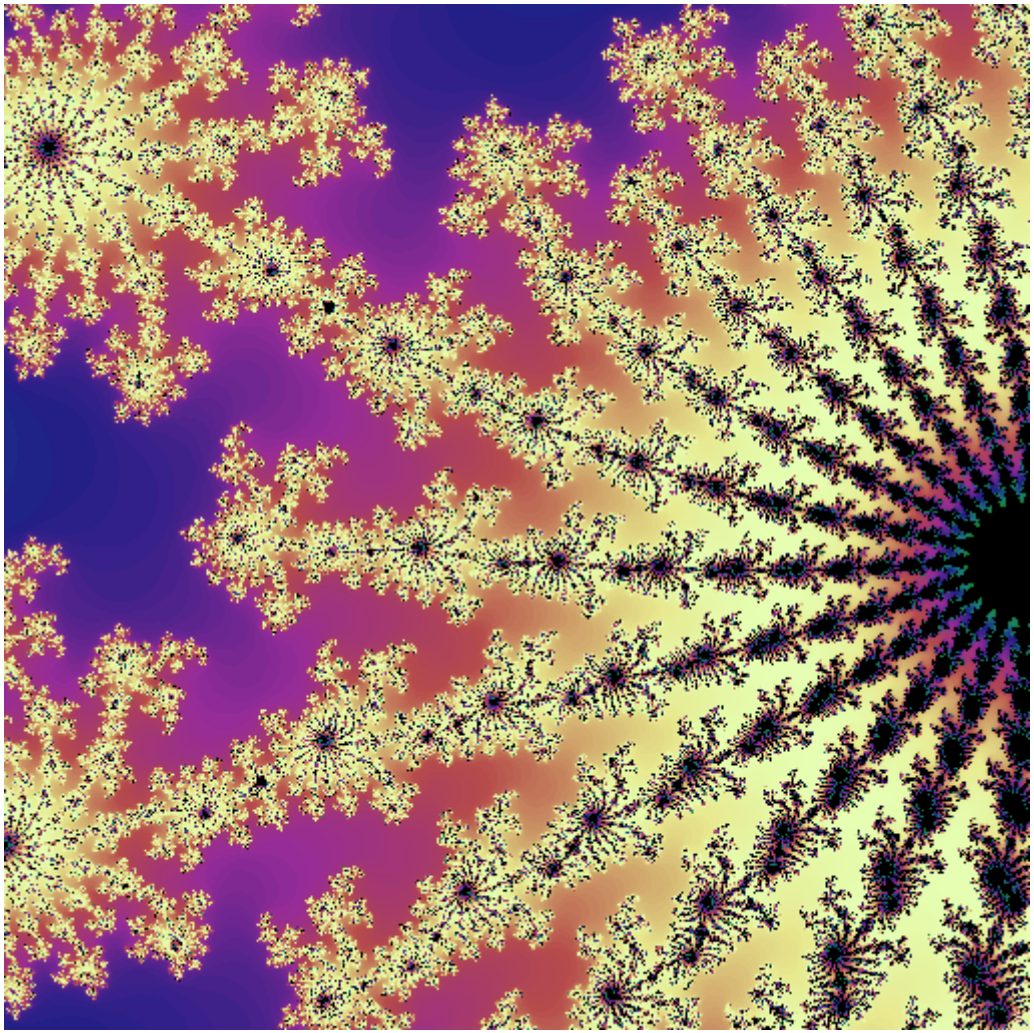
… mandelbrArt open now!

searching for Xanadu…



… mandelbrArt open now!

searching for Xanadu…



… mandelbrArt open now!

Abstraction with ADTs

implementation vs interface
interface: opaque values; details hidden from user
implementation: `struct` and function definitions

interface is a *well-defined boundary*…
implementation shouldn't trust users;
users shouldn't trust implementation

Abstraction with ADTs

implementation vs interface
interface: opaque values; details hidden from user

```
// in Complex.h:
typedef struct _complex *Complex;

Complex newComplex (double re, double im);
void destroyComplex (Complex c);
double complexRe (Complex c);
double complexIm (Complex c);
double complexMod (Complex c);
double complexArg (Complex c);
```

how does this ADT store values? (x, y)? (r, θ)?
we don't know, and don't need to know

Abstraction with ADTs

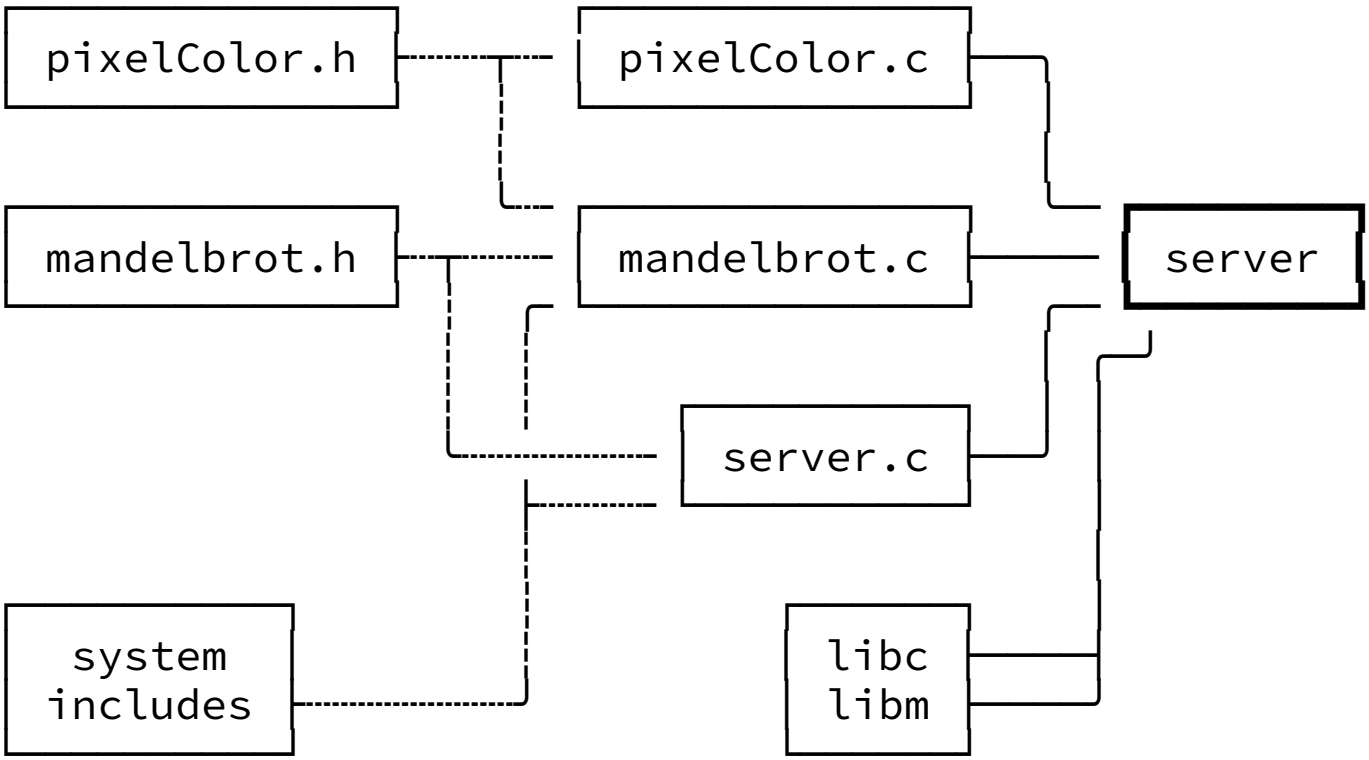
implementation vs interface

implementation: `struct` and function definitions

```
// in Complex.c:
typedef struct _complex {
    double re;
    double im;
} complex;

complex *newComplex (double re, double im) { /* ... */ }
void destroyComplex (void) { /* ... */ }
double complexRe (complex *c)  { /* ... */ }
double complexIm (complex *c)  { /* ... */ }
double complexMod (complex *c) { /* ... */ }
double complexArg (complex *c) { /* ... */ }
```

Aside: Linking it all together



ADT Jargon

interface
the header file

implementation
the .c file with functions defined

consumer, user
other .c files that use the functions

constructor
makes a new instance of the ADT

destructor
destroys an instance of the ADT

getter, setter
retrieve or change a value in an instance

… but how do we know that it works?

Testing.

Four Types of Testing

1. Bad tests

“I’ve written this program…
now, let’s write some tests…
my program passed, woohoo!”

Four Types of Testing

2. Black box tests

"I've written this program,
now let's get someone else
to test it for me!"

your program is
a magical black box,
where information goes in,
and information comes out.

Four Types of Testing

3. White box tests

"I've written this program,
can you look through it,
and check it's right?"

Four Types of Testing

4. Unit tests

“As well as testing my whole program,
I’ll test each of the small parts of it.”

faster and easier to check our small units
and then check the whole program

Testing with `assert`


```
#include <assert.h>

#include "Complex.h"
```

```
    Complex c = newComplex (0, 0);

    assert (c != NULL);
    assert (complexRe (c) == 0);
    assert (complexIm (c) == 0);

    destroyComplex (c);
```