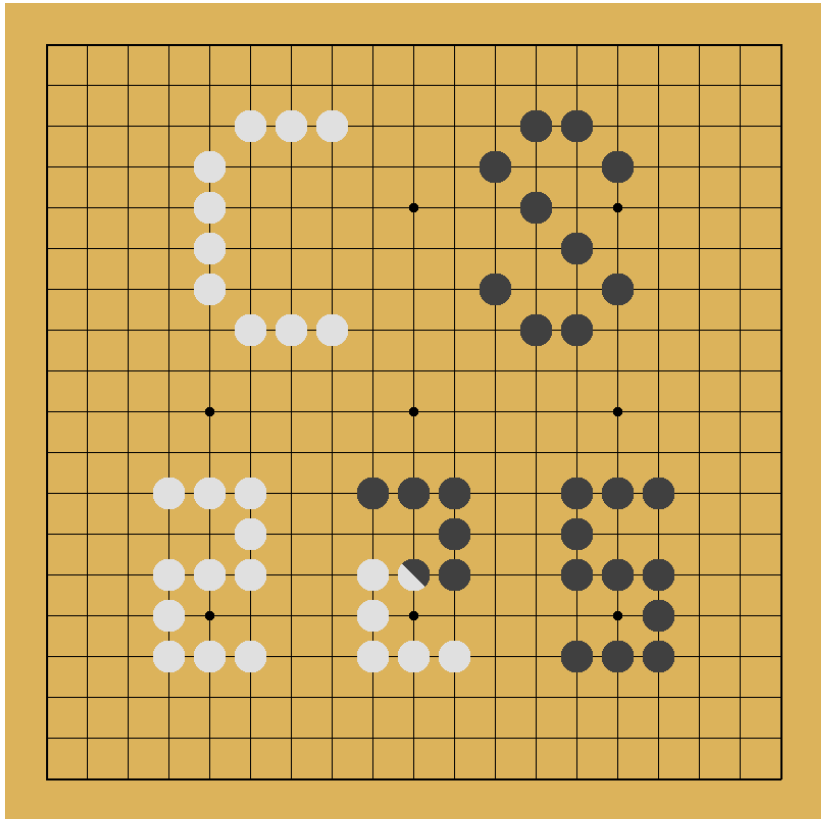


Data Structures

Feb. 2 – Templates

Wade Fagen-Ulmschneider

Polymorphism

Object-Orientated Programming (OOP) concept that a single object may take on the type of any of its base types.

Sphere.cpp

```
Sphere::print_1() {
  cout << "Sphere" << endl;
}

Sphere::print_2() {
  cout << "Sphere" << endl;
}

virtual Sphere::print_3() {
  cout << "Sphere" << endl;
}

virtual Sphere::print_4() {
  cout << "Sphere" << endl;
}

// In .h file:
virtual Sphere::print_5() = 0;
```

Planet.cpp

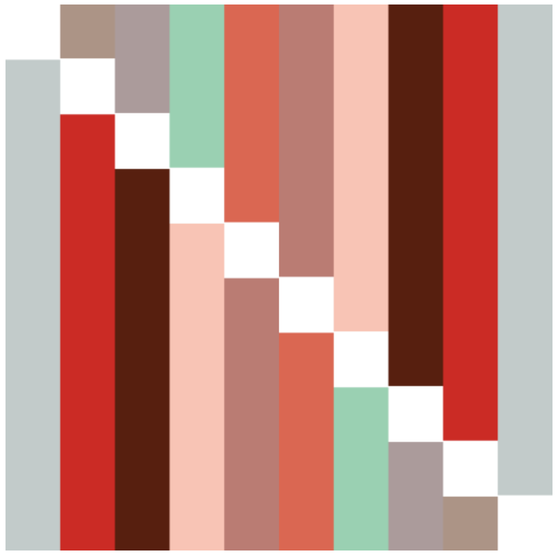
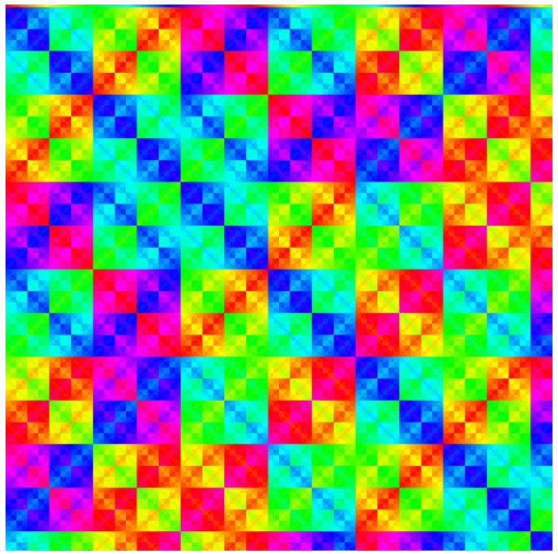
```
// No print_1() in RedBall.cpp




Planet::print_2() {
  cout << “Earth" << endl;
}

// No print_3() in RedBall.cpp



Planet::print_4() {
  cout << “Earth" << endl;
}

Planet::print_5() {
  cout << “Earth" << endl;
}
```

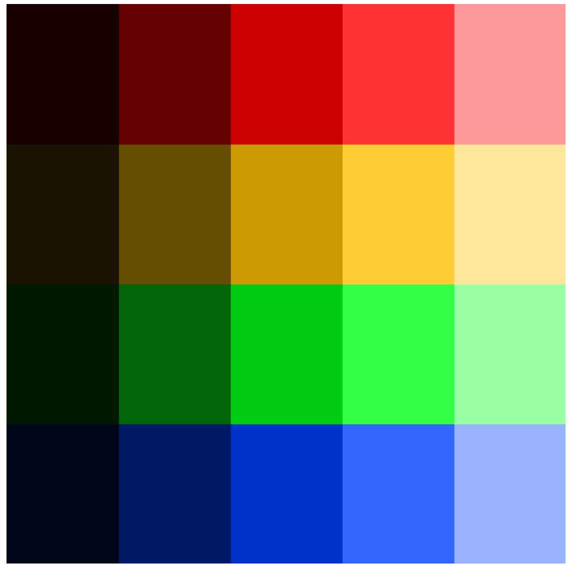
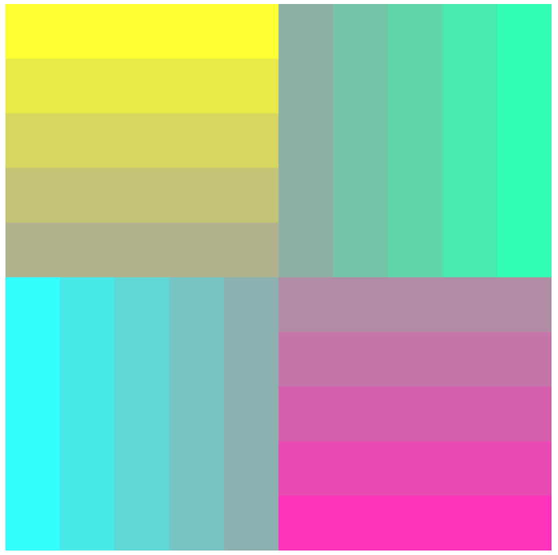
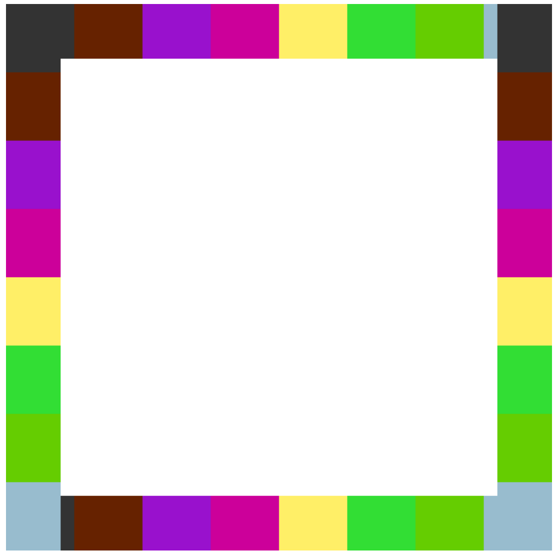
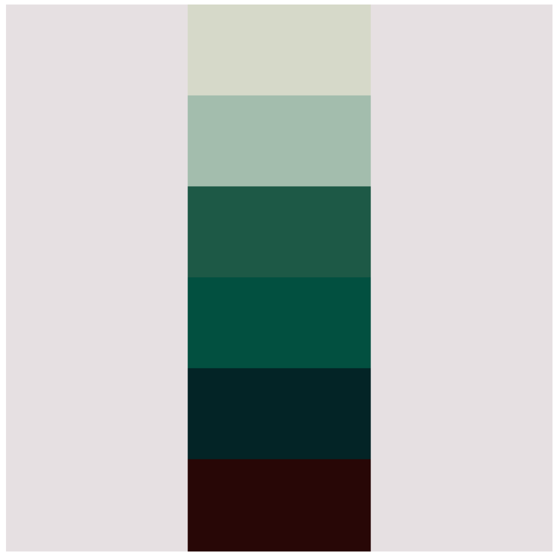
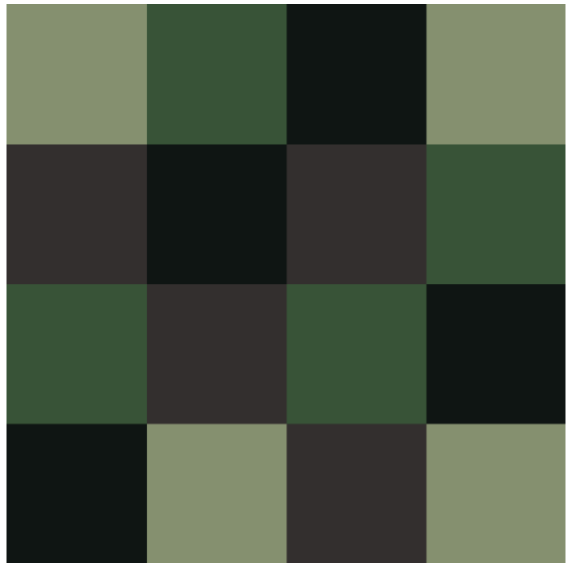
Runtime of Virtual Functions

	`Sphere obj;`	`Planet obj;`	`Planet r;` `Sphere &obj = r;`
`obj.print_1();`			
`obj.print_2();`			
`obj.print_3();`			
`obj.print_4();`			
`obj.print_5();`			

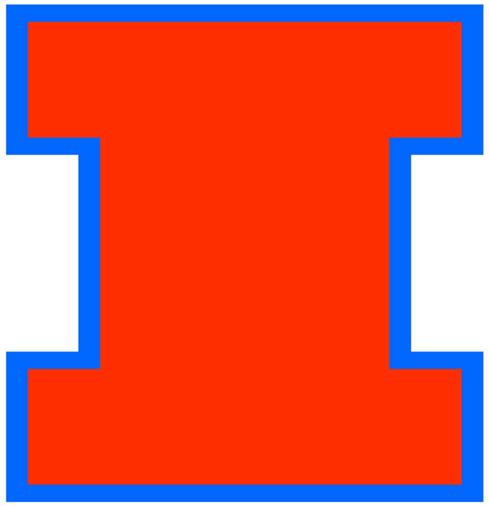
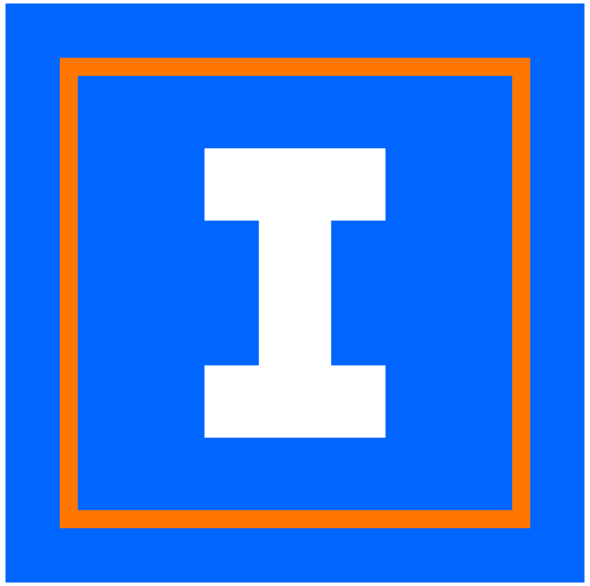
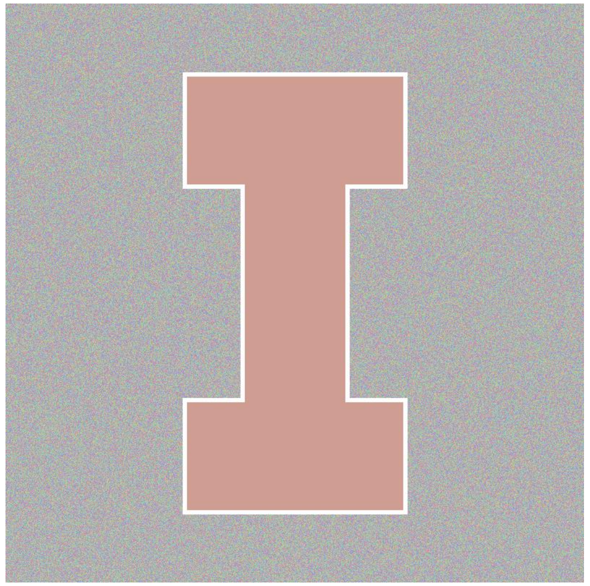
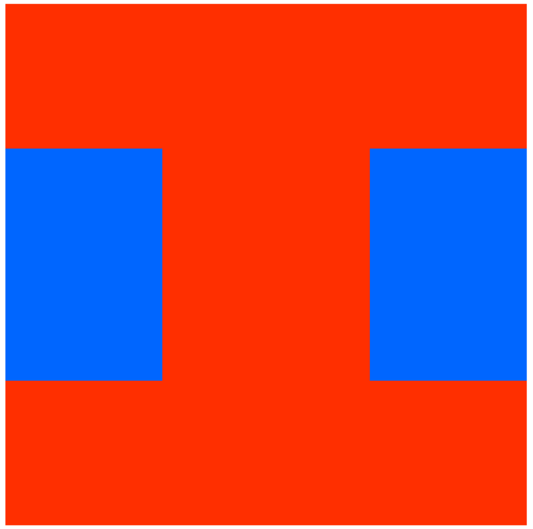
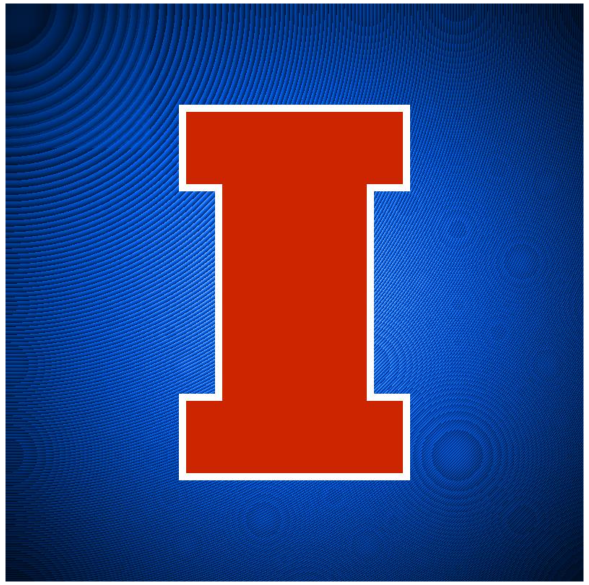
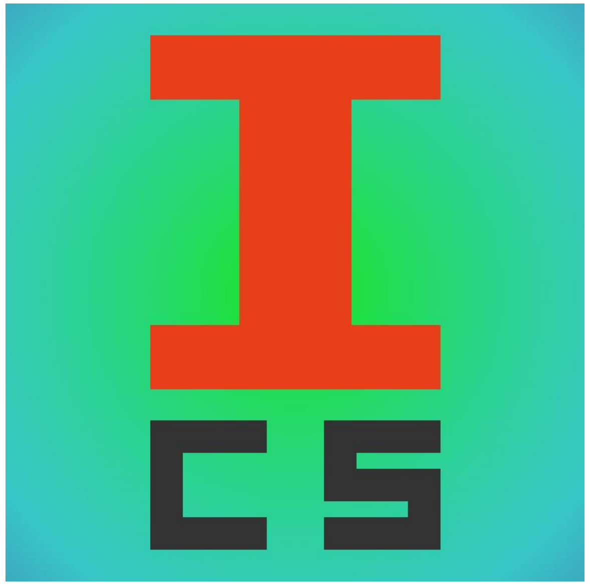
Why Polymorphism?

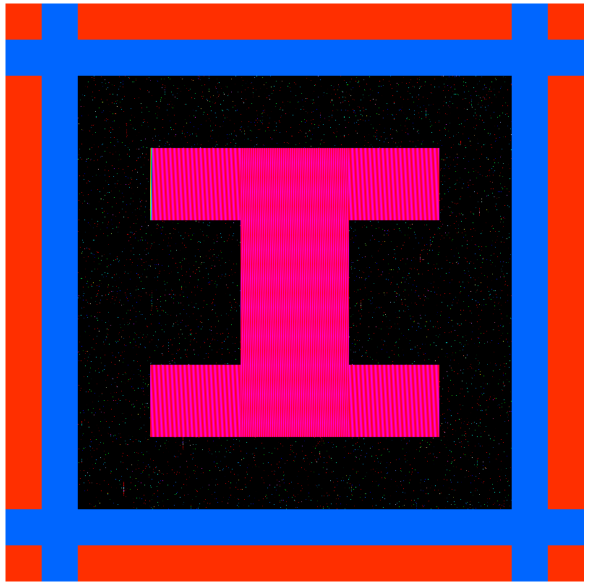
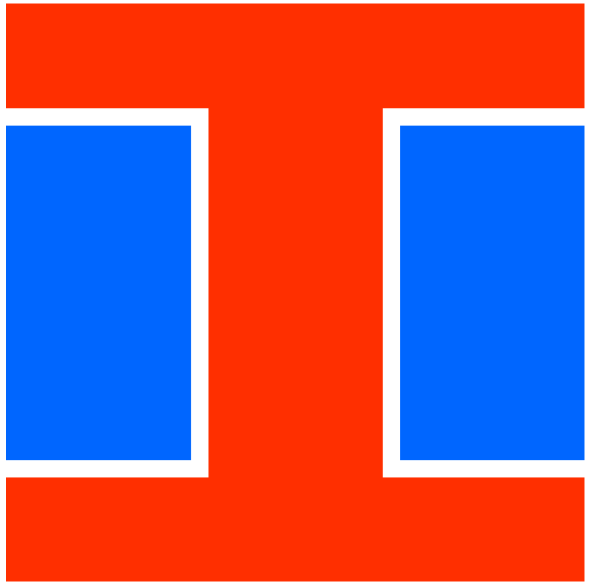
MP1 Artwork

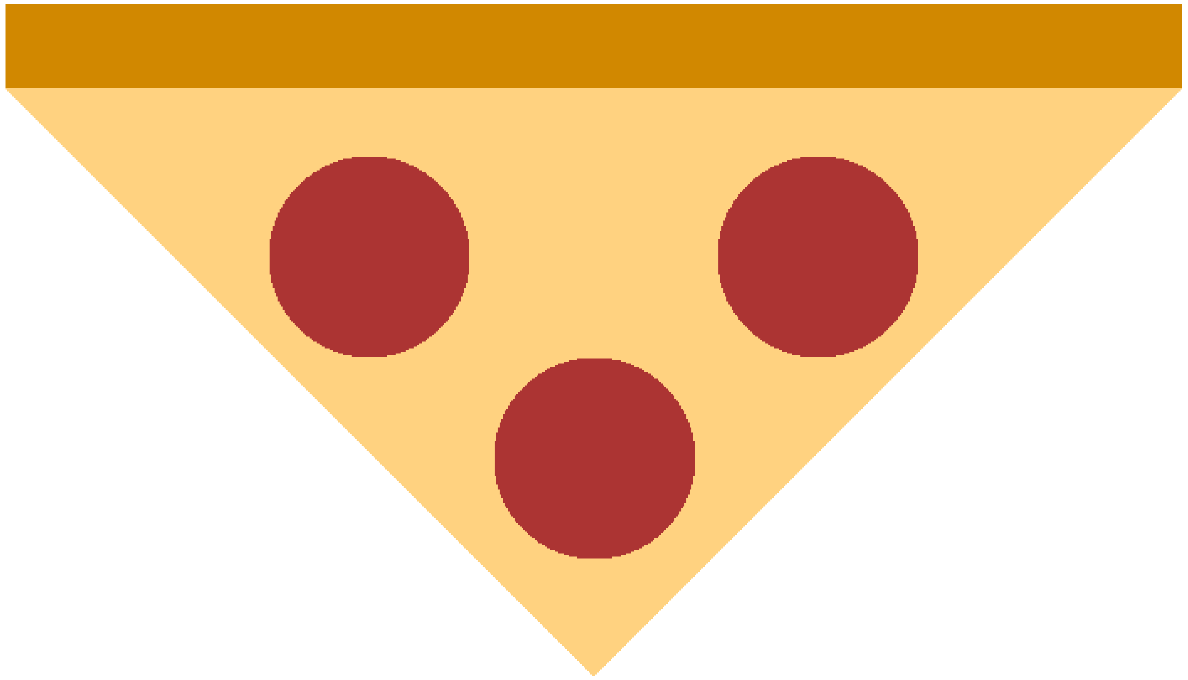



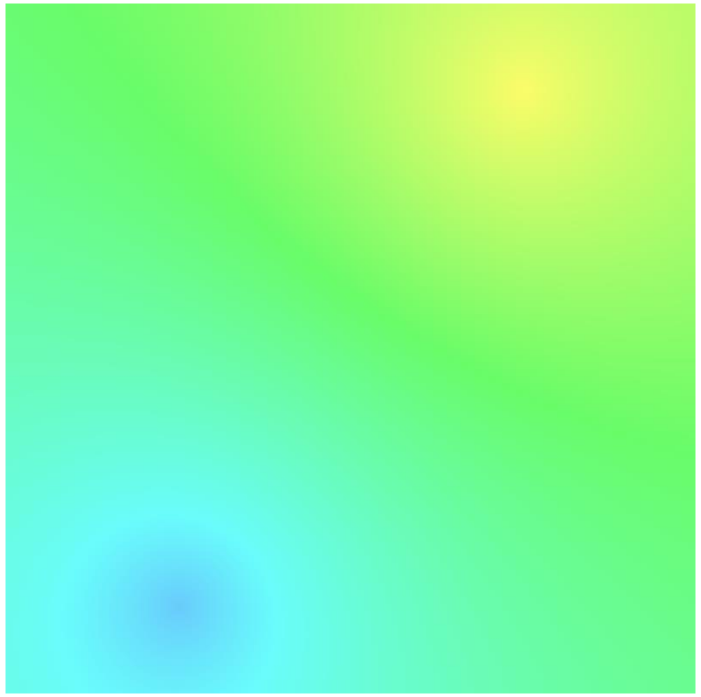
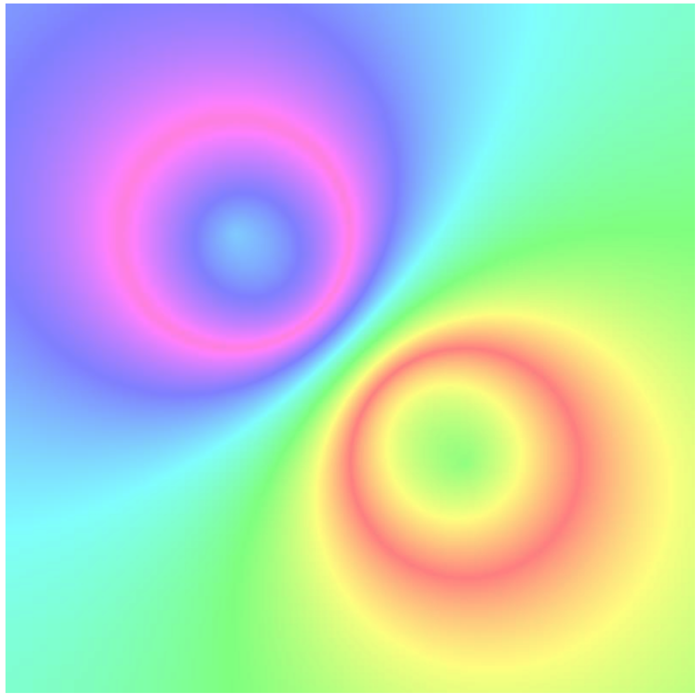






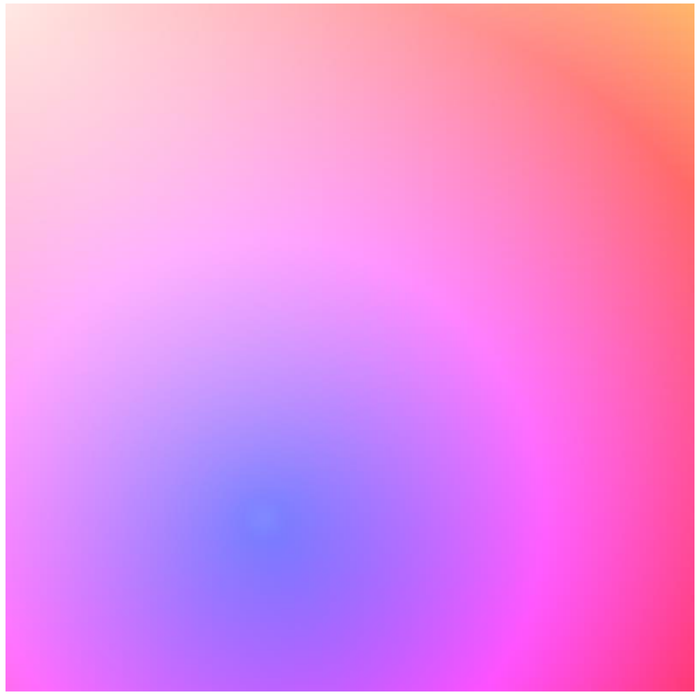
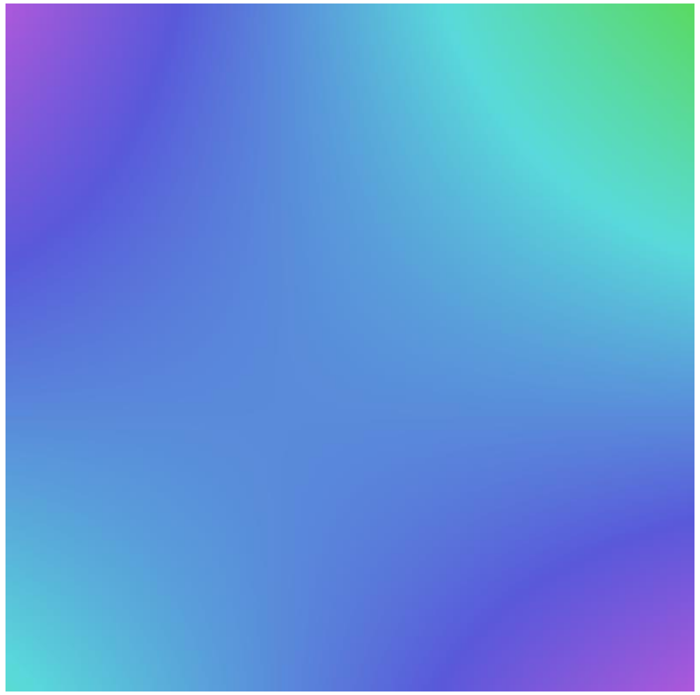
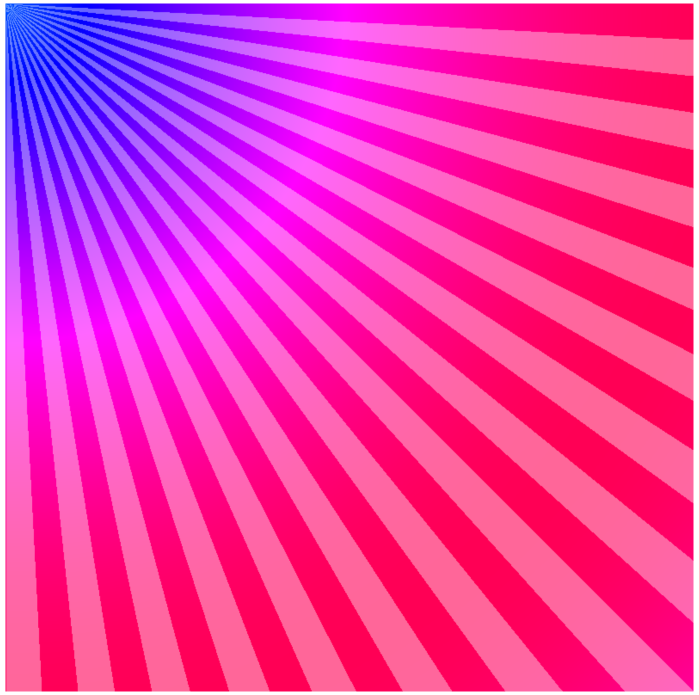
MP1 Artwork

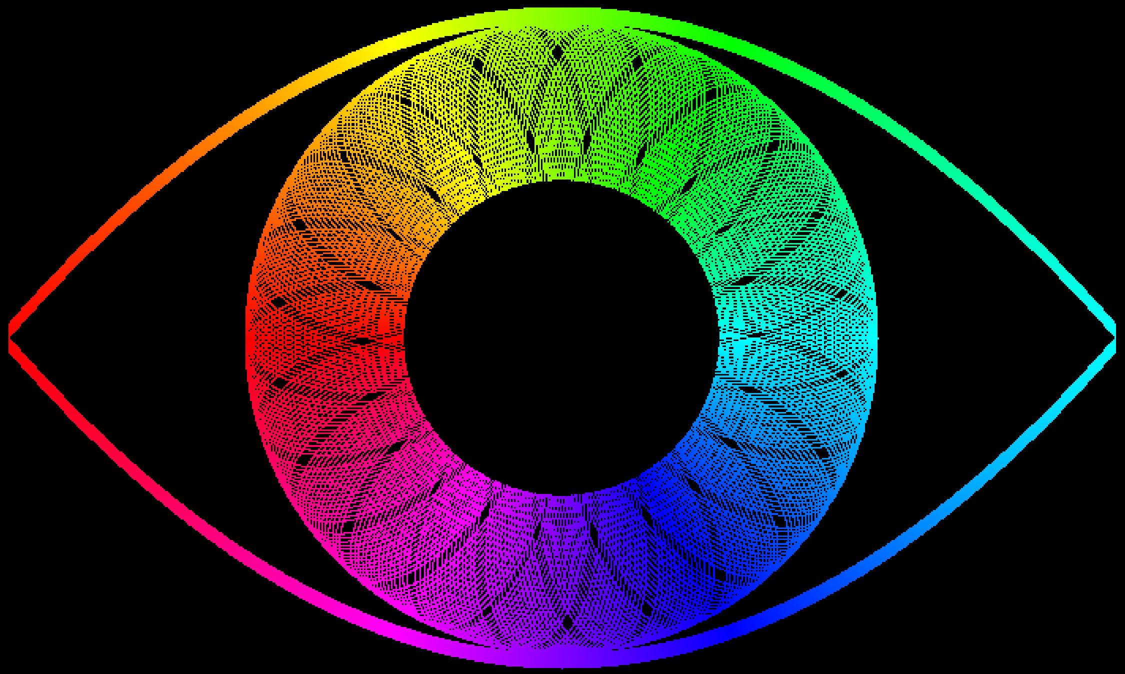
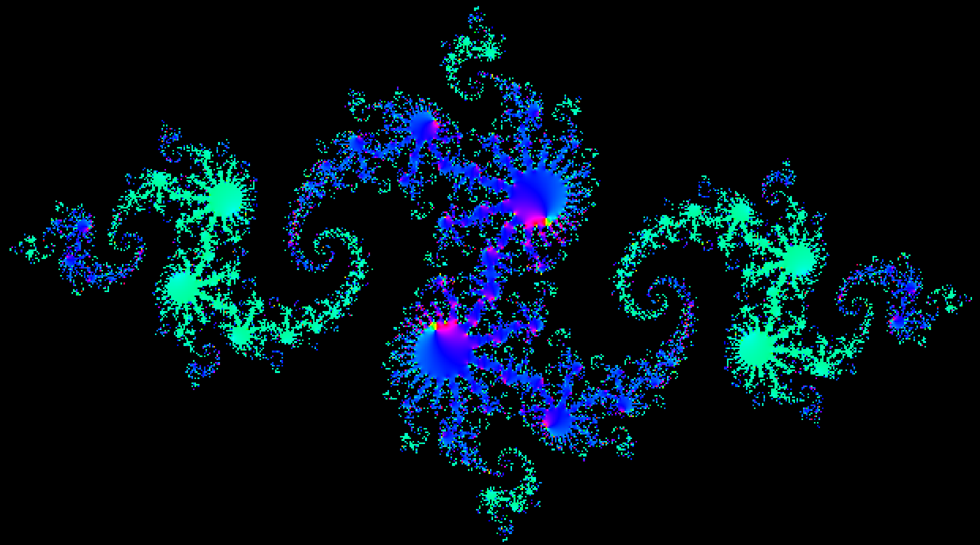







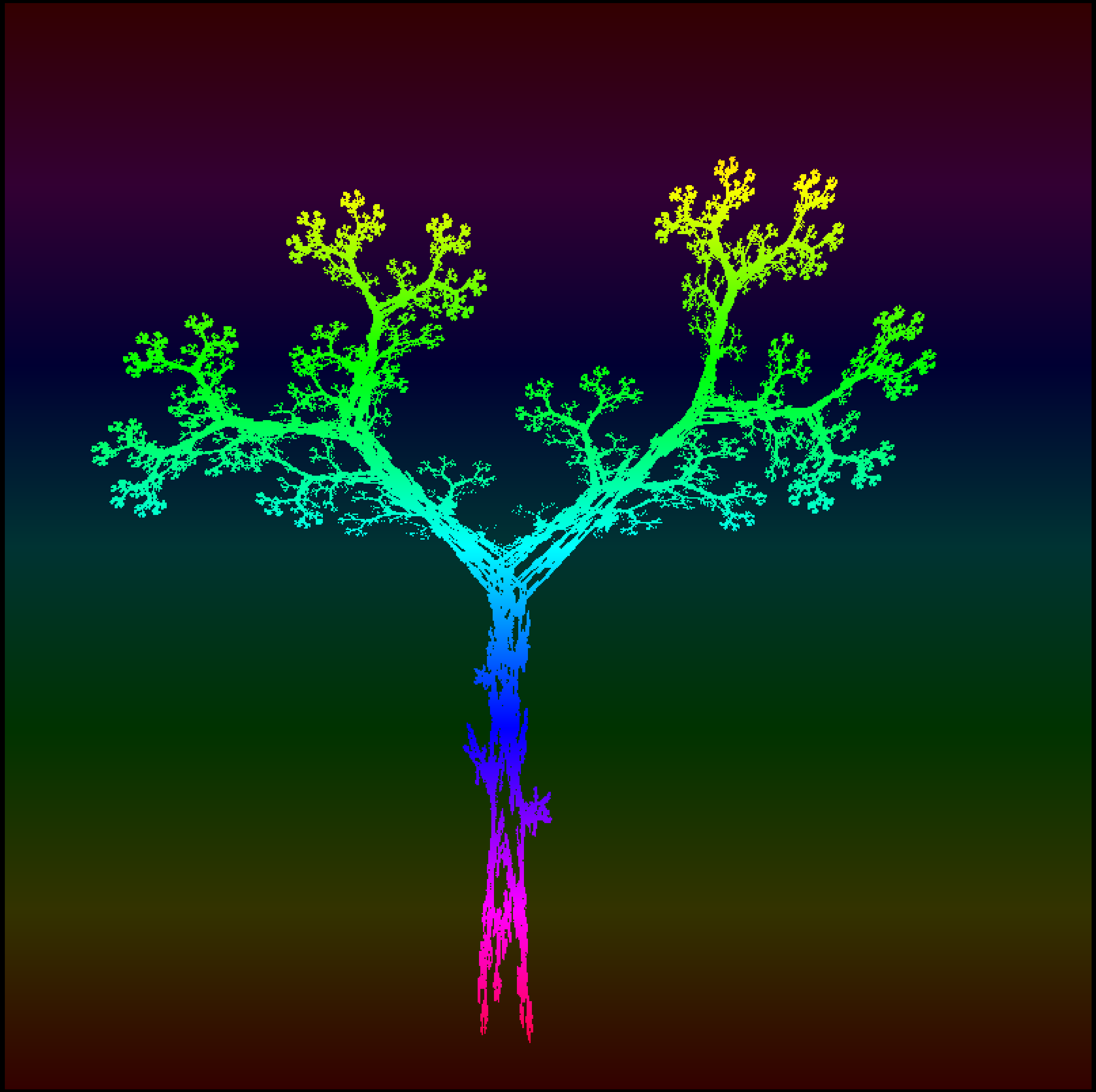
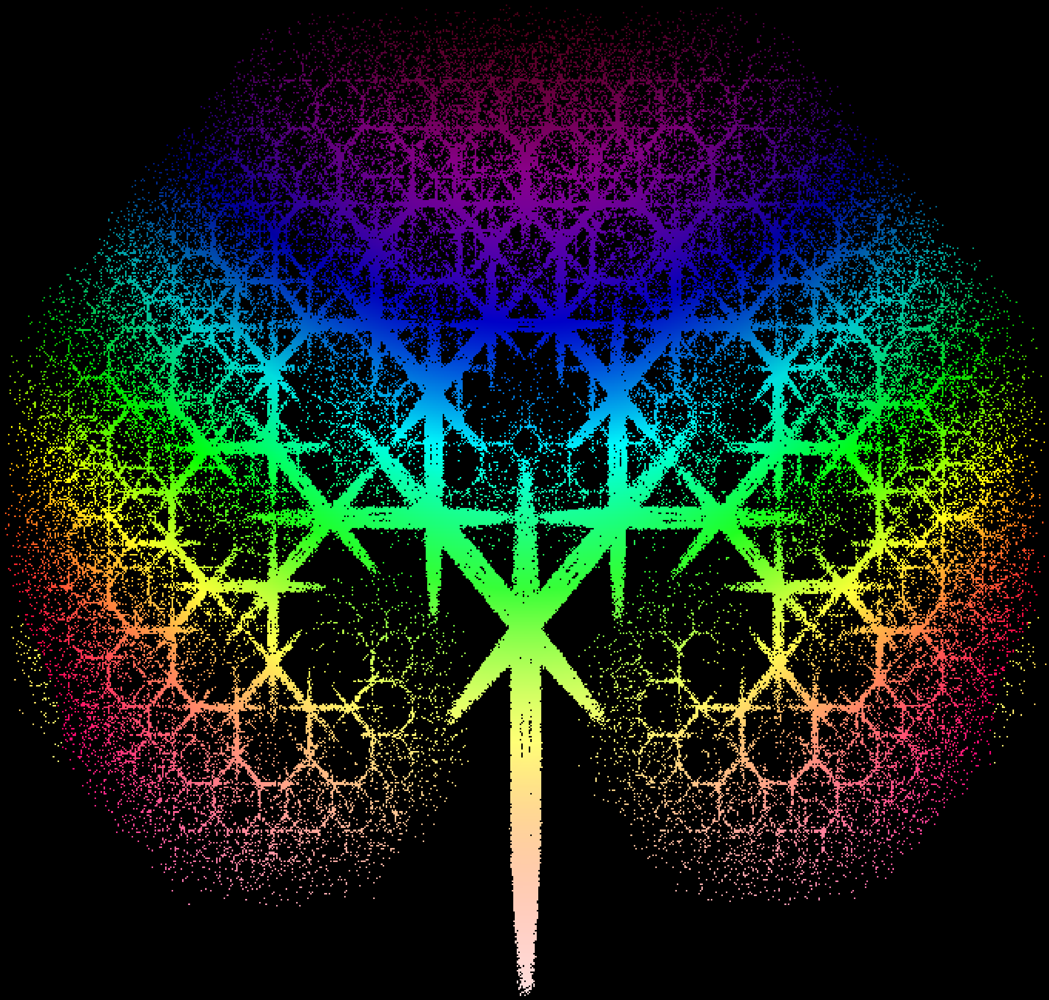



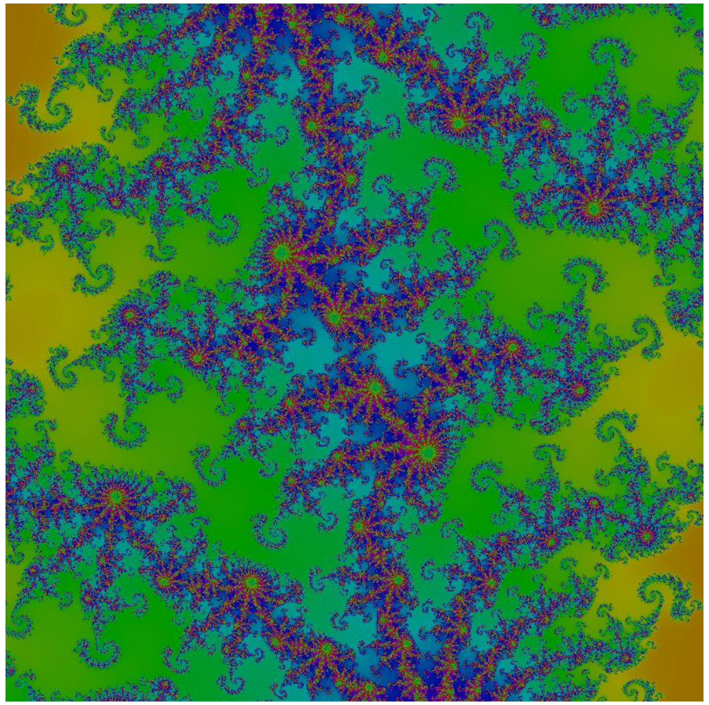
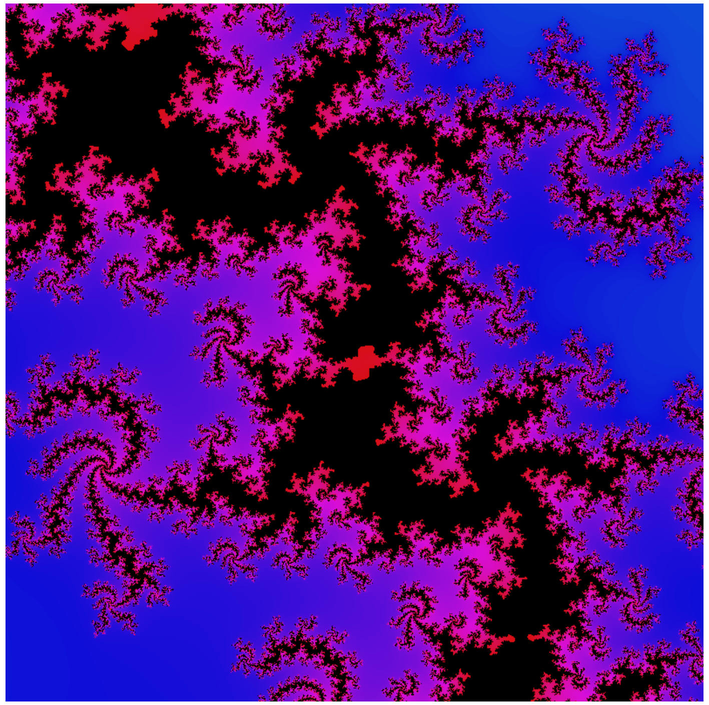
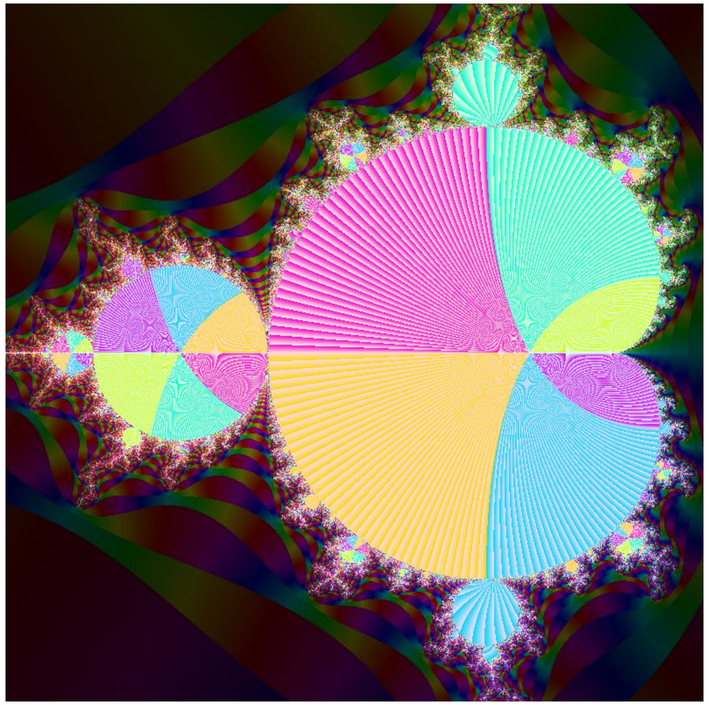
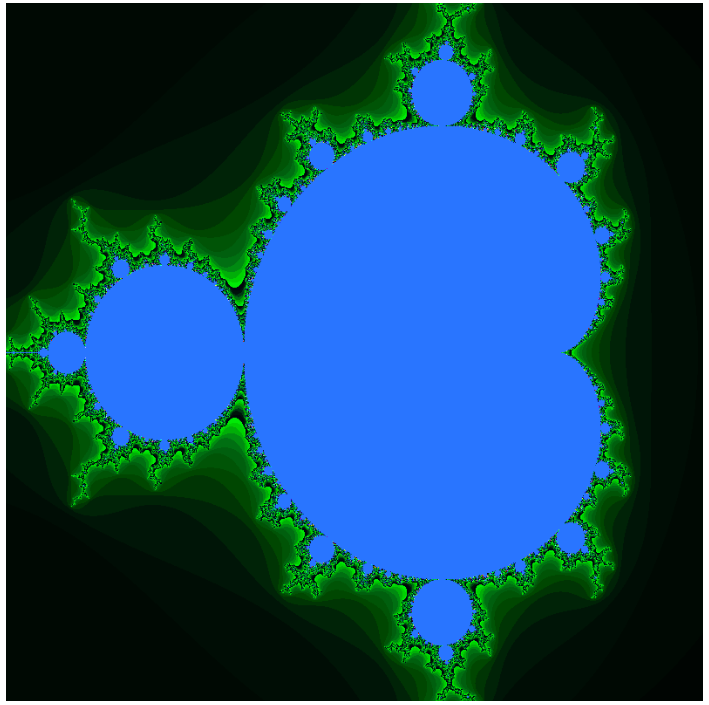
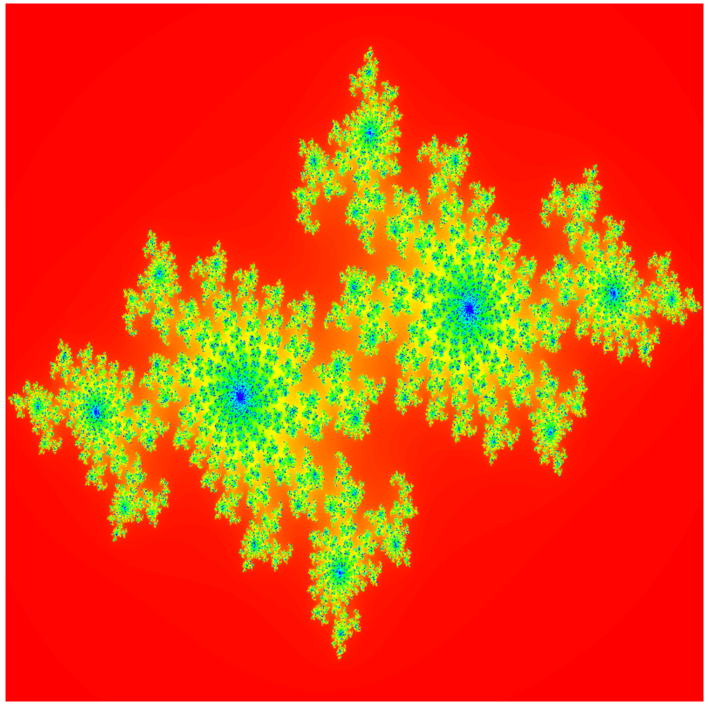
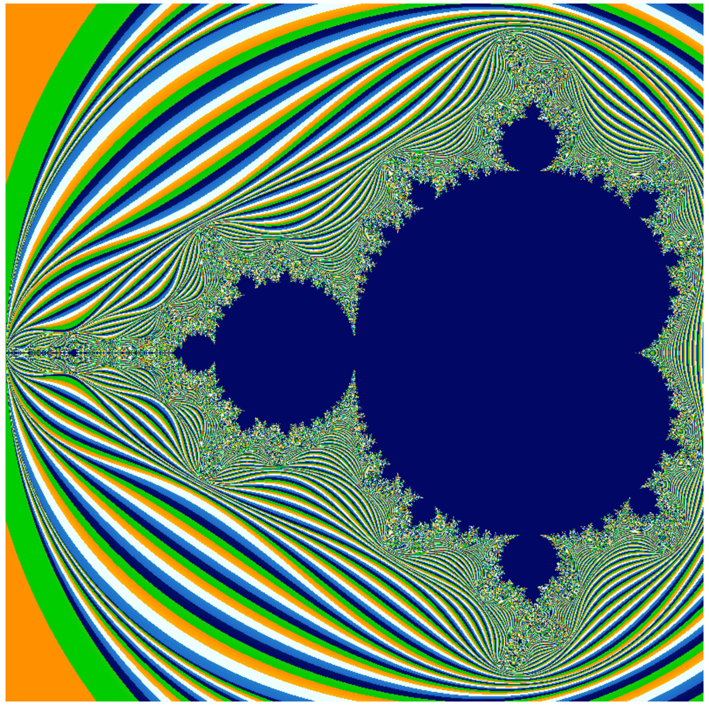


MP: Extra Credit

The most successful MP is an MP done early!

Unless otherwise specified in the MP, we will award +1 extra credit point per day **for completing Part 1** before the due date *(up to +7 points)*: Example for MP2:

+7 points: Complete by Monday, Feb. 5 (11:59pm)
+6 points: Complete by Tuesday, Feb. 6 (11:59pm)
+5 points: Complete by Wednesday, Feb. 7 (11:59pm)
+4 points: Complete by Thursday, Feb. 8 (11:59pm)
+3 points: Complete by Friday, Feb. 9 (11:59pm)
+2 points: Complete by Saturday, Feb. 10 (11:59pm)
+1 points: Complete by Sunday, Feb. 11 (11:59pm)
MP2 Due Date: Monday, Feb 12

MP: Extra Credit

The most successful MP is an MP done early!
We will give partial credit and maximize the value of your extra credit:

You made a submission and missed a few edge cases in Part 1:
Monday: +7 * 80% = **+5.6** earned

You fixed your code and got a perfect score on Part 1:
Tuesday: +6 * 100% = **+6** earned *(maximum benefit)*

You began working on Part 2, but added a seg fault to Part 1:
Wednesday: +5 * 0% = +0 earned *(okay to score lower later)*

…

animalShelter.cpp

```
class Animal {
  public:
    void speak() {                                    }
};

class Dog : public Sphere {
  public:
     void speak() {                                    }
};

class Cat : public Sphere {
  public:
     void speak() {                                    }
};
```

Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:

virtual-dtor.cpp

```
class Sphere {
  public:
    ~Sphere();
};

class Planet : public Sphere {
  public:
    ~Sphere();
};
```

Assignment Operator

sphere.h

```cpp
class Sphere {
  public:
    Sphere();
    Sphere(double r);
    Sphere(const Sphere & other);
    ~Sphere();

    Sphere & operator=(Sphere & other);

    double getRadius() const;
    double getVolume() const;

    std::string[] getProps() const;
    void addProp(std::string prop);

  private:
    double r_;
    std::string * props_;
    unsigned props_max_, props_ct_;
    void _destroy();
    void _copy(Sphere & other);
};
```

assignmentOpSelf.cpp

```cpp
#include "Sphere.h"

int main() {
  cs225::Sphere s(10);
  s = s;
  return 0;
}
```

sphere.cpp

```cpp
void Sphere::_destroy() {  delete[] props_;   }

void Sphere::_copy(const Sphere &other) {
  r_ = other.r;
  props_max_ = other.props_max_;
  props_ct_ = other.props_ct_;
  props_ = new std::string[10];
  for (unsigned i = 0; i < props_ct_; i++) {
    props_[i] = other.props_[i];
  }
}

Sphere& Sphere::operator=(const Sphere &other) {



  _destroy();
  _copy(other);
  return *this;
}
```

assignmentOpSelf.cpp

```cpp
#include "Sphere.h"

int main() {
  cs225::Sphere s(10);
  s = s;
  return 0;
}
```

cs225/png.h

```cpp
class PNG {
  public:
    PNG();
    PNG(unsigned int width, unsigned int height);
    PNG(PNG const & other);
    ~PNG();

    PNG & operator= (PNG const & other);
    bool operator== (PNG const & other) const;

    bool readFromFile(string const & fileName);
    bool writeToFile(string const & fileName);
    HSLAPixel & getPixel(unsigned int x, unsigned int y) const;
    unsigned int width() const;
    // ...

  private:
    unsigned int width_;
    unsigned int height_;
    HSLAPixel *imageData_;
    void _copy(PNG const & other);
};
```

Abstract Data Type

List ADT

What types of “stuff” do we want in our list?

Templates

template1.cpp

```


T maximum(T a, T b) {
  T result;
  result = (a > b) ? a : b;
  return result;
}
```

List.h

```
#ifndef LIST_H_
#define LIST_H_



class List {
  public:








  private:



};

#endif
```

List.cpp

```
```