Full Name: _____

Student ID#: _____

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Exam #3 «---» R 001||002||003 42 «---» Exam #3

Instructions

1. Students are allowed pencils, erasers, and beverage only.
2. All books, bags, backpacks, phones, **smart watches**, and other electronics, etc. must be placed along the walls. **Silence all notifications.**
3. PRINT your name and student ID at the top of this page **and every coding sheet**, and verify that you have all pages.
4. **Read every question completely before you start to answer it.** If you have a question, please raise your hand. You may or may not get an answer, but it won't hurt to ask.
5. If you leave the room, you may not return.
6. You are required to SIGN and ABIDE BY the following Honor Pledge for each exam this semester.

NOTE: The number of questions in each section, and the topic of Free Response questions, may vary on the actual Final Exam.

Honor Pledge

On my honor, I pledge that I will not attempt to communicate with another student, view another student's work, or view any unauthorized notes or electronic devices during this exam. I understand that the professor and the CSE 1325 Course Curriculum Committee have zero tolerance for cheating of any kind, and that any violation of this pledge or the University honor code will result in an automatic grade of zero for the semester and referral to the Office of Student Conduct for scholastic dishonesty.

Student Signature: _____

WARNING: Questions are on the BACK of this page!

Vocabulary

Write the word or phrase from the Words list below to the left of the definition that it best matches. Each word or phrase is used at most once, but some will not be used. {15 at 2 points each}

Exam 1

VOCAB KEY

1 Iterator

2 Definition

3 Abstract Method

4 Multiple Inheritance

5 Destructor

6 Field

7 Operator Overloading

8 Container

9 Exception

10 Operator

MULTIPLE CHOICE KEY

1 D	6 D	11 C
2 D	7 A	12 A
3 D	8 B	13 C
4 B	9 D	14 C
5 A	10 C	15 B

Free Response

Provide clear, concise answers to each question. Write only the code that is requested. You will NOT write an entire application! You need NOT copy any code provided to you - just write the additional code specified. You need NOT write `#include` statements - assume you have what you need. You will write a .h guard only once (question 2.b) - skip them on all other .h files to save time.

While multiple questions may relate to a given application or class diagram, **each question is fully independent and may be solved as a stand-alone problem.** Thus, if you aren't able to solve a question, skip it until the end and move on to the next.

1. {10 points} (file I/O, map, find, arguments) Text file data formats list `city, state:population` on each newline-terminated row, like this.

```
Arlington, TX:395394
Arlington, VA:234965
Ottumwa, IA:24454
```

Output from the program you will write below, if correct, will look something like this.

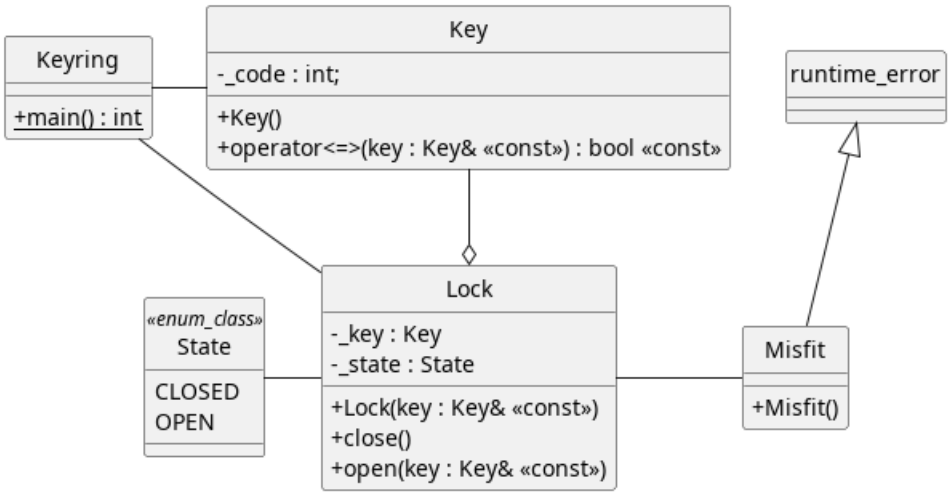
```
Enter city, ST: Arlington, TX
Arlington, TX pop is 398431
Enter city, ST: Vicksburg, MS
Vicksburg, MS not found
Enter city, ST: ^D
```

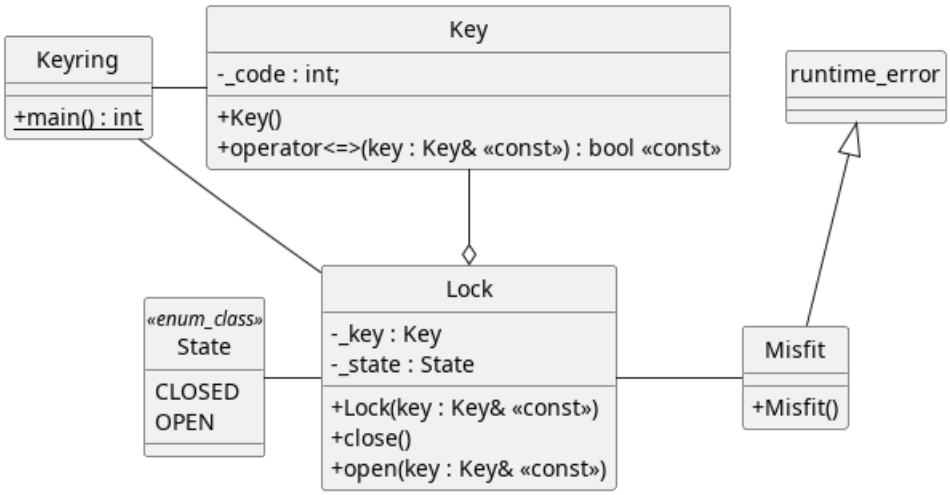
Write a main function that performs lookup on this data by following the comment prompts. Assume all necessary libraries have already been included.

```
// Define type Location to be a string and Population to be an int.

typedef std::string Location;
typedef int Population;

// Begin a main function. Accept command line arguments.

int main(int argc, char* argv[]) { // OR
int main(int argc, char** argv) {

// Define a map using Location as the key and Population as the value.

std::map<Location, Population> pop_map;

// Open the first argument as the filename of the above data file for input.
// You may assume this succeeds.

std::ifstream ifs{std::string{argv[1]}};
```

```
// Loop, reading the data from each line of text in the data file as a Location and
// a Population, adding each pair to the map.

Location location;
Population population;
while(std::getline(ifs, location, ':')) {
    ifs >> population; ifs.ignore();
    pop_map[location] = population;
}

// After the loop ends, if the end of the data file had not been reached
// when the read loop completed, stream "Error reading " and the filename
// to the standard error output and return error code -1 to the operating system.

if(!ifs.eof()) {
    std::cerr << "Error reading " << argv[1] << std::endl;
    return -1; // OR exit(-1);
}

// Print "Enter city, ST: " to the console

std::cout << "Enter city, ST: ";

//Read a Location from the keyboard.

std::getline(std::cin, location);

// Use map's find method to obtain an iterator to the key/value pair for the
// entered location. If found, print the Location followed by " pop is " and
// its Population. Remember that key/value pairs in a C++ map are accessed
// as public fields first (for the key) and second (for the value).
// If not found, print Location followed by " not found".

auto it = pop_map.find(location);
if(it != pop_map.end())
    std::cout << location // OR it->first
              << " pop is " <<  it->second << std::endl;
else
    std::cerr << location << " not found" << std::endl;

// (end of question 1)
```

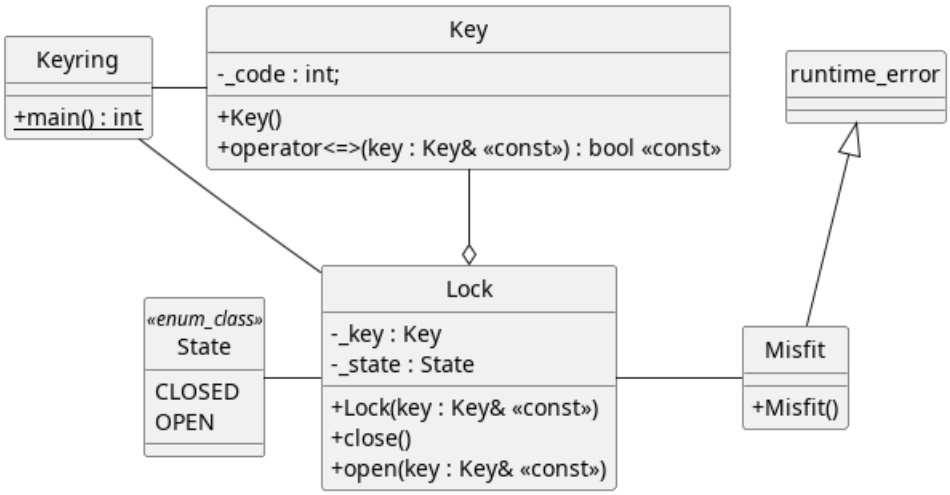
2. (enum, custom exception, try/catch, op overloading, vector, sort) Consider the following C++ class declaration and main function. Guard and includes are omitted.





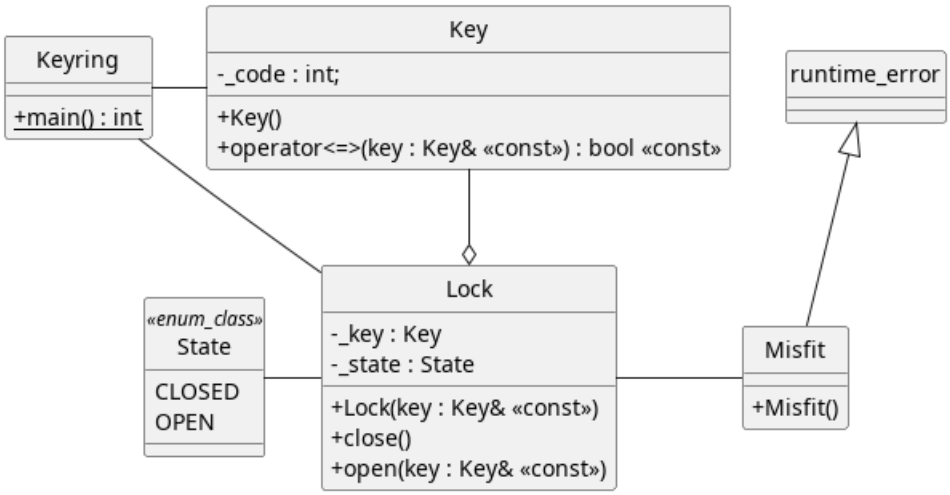
a. {4 points} In file state.h, write enum class `State`.

```
enum class State{CLOSED, OPEN};
```



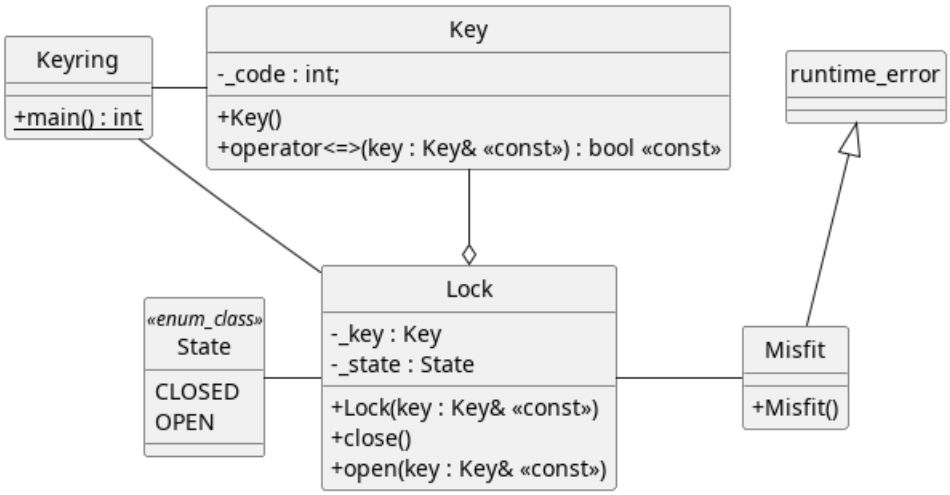
b. {5 points} In file misfit.h, write exception `Misfit` which inherits from `std::runtime_error`. DO include a guard here, but skip them for the rest of the exam. Do NOT write `#include` here or elsewhere - assume you have whatever you need.

```
#ifndef __MISFIT_H
#define __MISFIT_H

class Misfit : public std::runtime_error {
  public:
    Misfit();
};

#endif
```



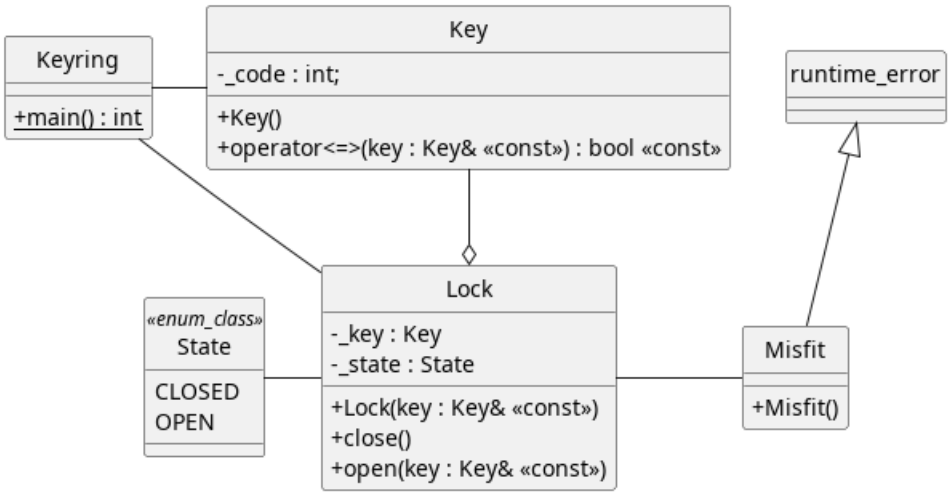
c. {3 points} In file misfit.cpp, write the constructor. Chain to the superclass constructor with the message "Wrong key for lock" as the parameter.

```
// Not required for question
#include "misfit.h"

const std::string msg {"Wrong key for lock"};
Misfit::Misfit() : std::runtime_error{msg} { }
```



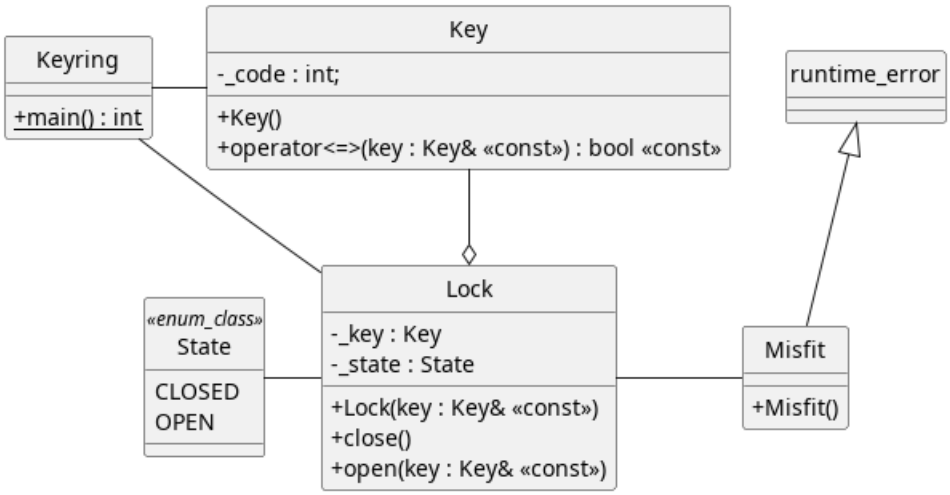
d. {3 points} In file key.h, declare all of class Key (omitting the guard and includes). The default <=> spaceship is *highly* recommended, or write the six operators ==, !=, <, <=, >, >= instead.

```
class Key {
  public:
    Key();
    auto operator<=>(const Key&) const = default;
  private:
    int _code;
};
```



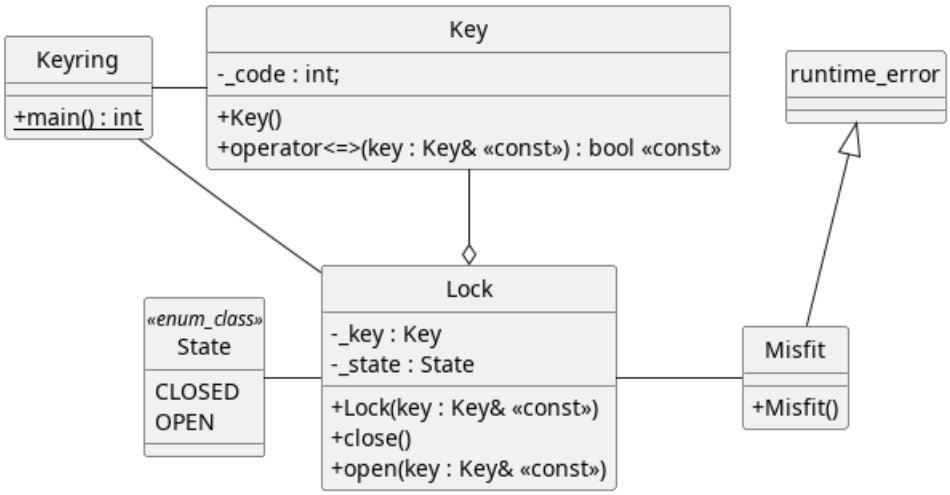
d. {3 points} In file key.h, declare all of class Key (omitting the guard and includes). The default <=> spaceship is *highly* recommended, or write the six operators ==, !=, <, <=, >, >= instead.

```
// OR

class Key {
  public:
    Key();
    inline bool operator==(const Key& rhs) const {return (compare(rhs) == 0);}
    inline bool operator!=(const Key& rhs) const {return (compare(rhs) != 0);}
    inline bool operator< (const Key& rhs) const {return (compare(rhs) <  0);}
    inline bool operator<=(const Key& rhs) const {return (compare(rhs) <= 0);}
    inline bool operator> (const Key& rhs) const {return (compare(rhs) >  0);}
    inline bool operator>=(const Key& rhs) const {return (compare(rhs) >= 0);}
  private:
    int _code;
    int compare(const Key& key) const;
};
```



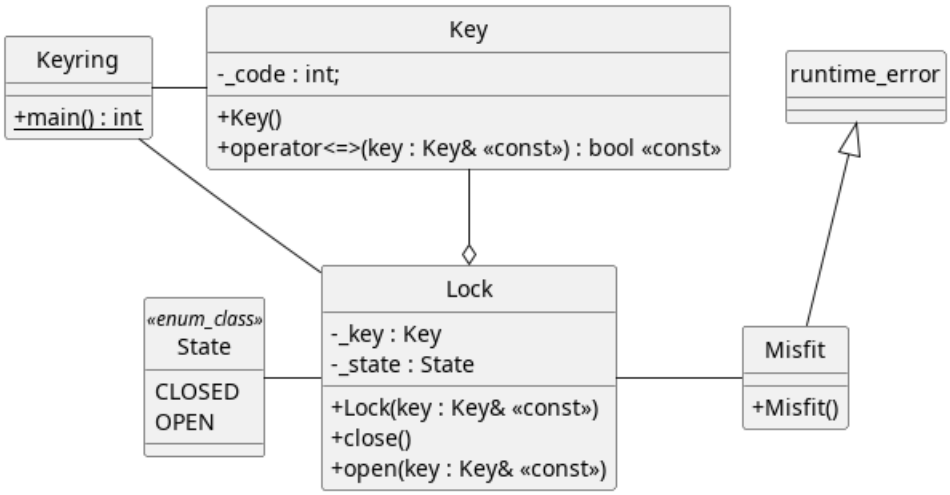
d. {3 points} In file key.h, declare all of class Key (omitting the guard and includes). The default <=> spaceship is *highly* recommended, or write the six operators ==, !=, <, <=, >, >= instead.

```
// OR

 class Key {
  public:
    Key();
#ifdef __cpp_impl_three_way_comparison
   auto operator<=>(const Key&) const = default;
#else
   inline bool operator==(const Key& rhs) const {return (compare(rhs) == 0);}
   inline bool operator!=(const Key& rhs) const {return (compare(rhs) != 0);}
   inline bool operator< (const Key& rhs) const {return (compare(rhs) <  0);}
   inline bool operator<=(const Key& rhs) const {return (compare(rhs) <= 0);}
   inline bool operator> (const Key& rhs) const {return (compare(rhs) >  0);}
   inline bool operator>=(const Key& rhs) const {return (compare(rhs) >= 0);}
#endif
 private:
   int _code;

#ifndef __cpp_impl_three_way_comparison
   int compare(const Key& key) const;
#endif
};
```



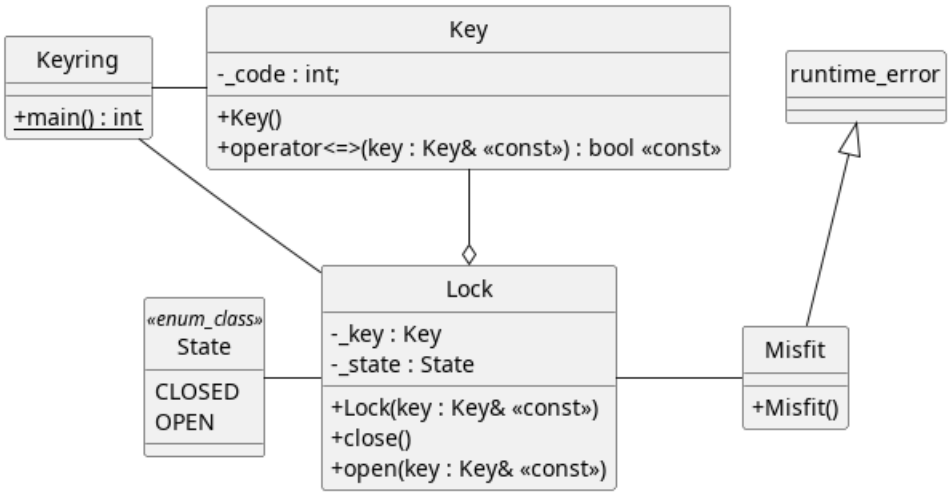
e. {3 points} In file lock.cpp, implement only method `open` for class `Lock`. The parameter is passed by const reference as shown in the class diagram. If the parameter `key` doesn't match the `_key` field, throw a `Misfit` exception. Otherwise, set field `_state` to `OPEN`.

```
void Lock::open(const Key& key) {
   if(key != _key) throw Misfit{};
   _state = State::OPEN;
}
```



`f.` {10 points} In file keyring.cpp, write the main function.

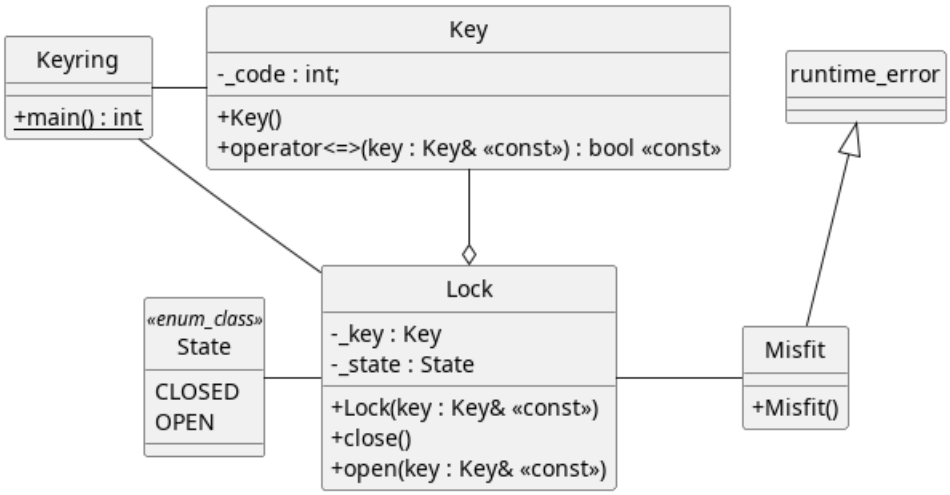
- Create a vector of Key objects named `keys` and a vector of Lock objects named `locks`. Add 10 `Key` objects to `keys`, and use those `Key` objects to construct 10 corresponding `Lock` objects for `locks`.
- Sort `keys`.
- Ask the user for a lock index in `locks` between 0 and 9, inclusive.
- Iterate over `keys`, trying each `Key` object in the lock object's `open` method. If the key doesn't fit, `open` will throw a `Misfit` exception, which should be ignored. But if no exception is thrown, the key fits and the lock is now open - print the index of that key that unlocked the selected lock.



`f.` {10 points} In file keyring.cpp, write the main function.

- Create a vector of Key objects named `keys` and a vector of Lock objects named `locks`.

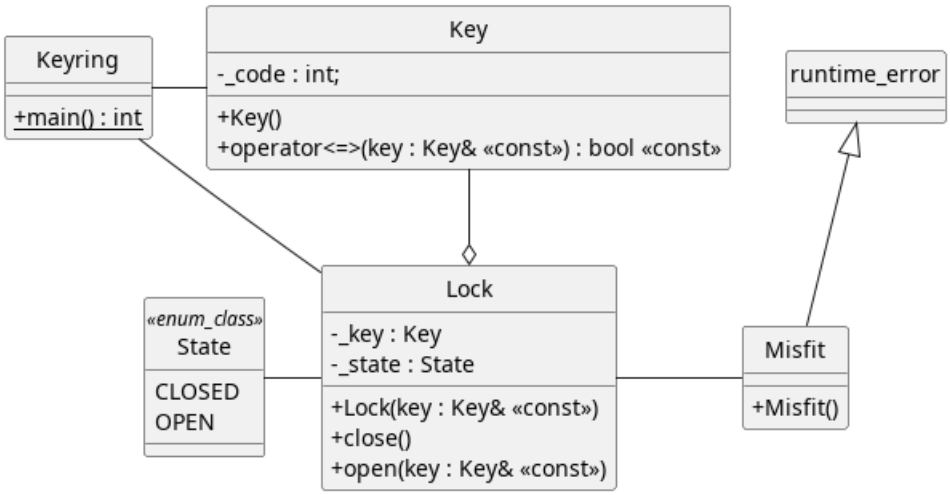
```
// Note: const is NOT required
const int NUM_LOCKS = 10;

int main() {
   srand(time(NULL)); // NOT required;

   // Note: Reserves are NOT required
   // Create a vector of Key objects named ``keys``
   std::vector<Key> keys;    keys.reserve(NUM_LOCKS);
   //     and a vector of Lock objects named ``locks``
   std::vector<Lock> locks;  locks.reserve(NUM_LOCKS);
```



`f.` {8 points} In file keyring.cpp, write the main function.

- Add 10 `Key` objects to `keys`, and use those `Key` objects to construct 10 corresponding `Lock` objects for `locks`.

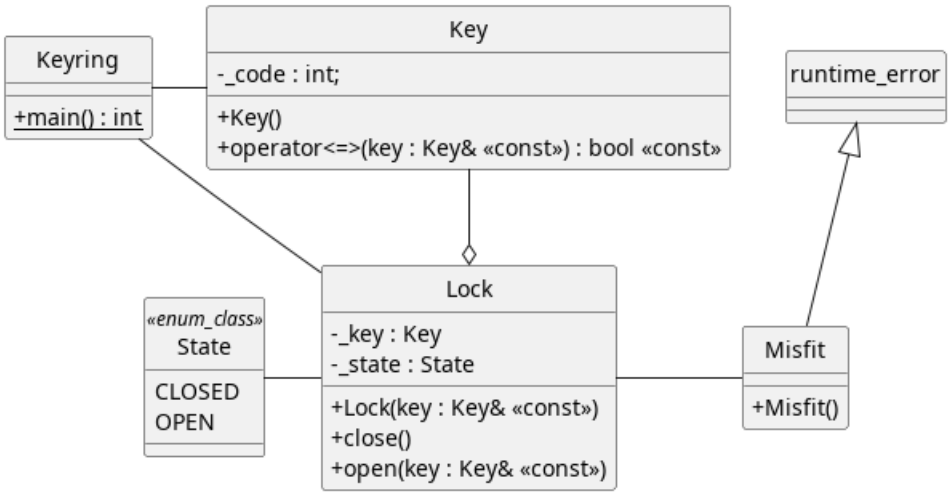
```
// Add 10 ``Key`` objects to ``keys``, and use those ``Key`` objects
// to construct 10 corresponding ``Lock`` objects for ``locks``
for(int i=0; i<NUM_LOCKS; ++i) {
    keys.push_back(Key{});
    locks.push_back(Lock{keys.back()});
}
```



`f.` {10 points} In file keyring.cpp, write the main function.

- Sort `keys`.

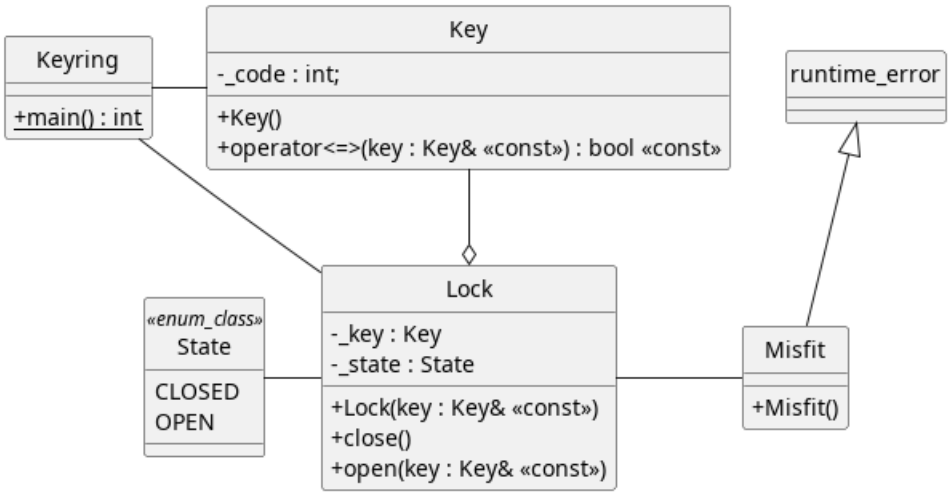
```
// Sort ``keys``
std::sort(keys.begin(), keys.end());
```



`f.` {10 points} In file keyring.cpp, write the main function.

- Ask the user for a lock index in `locks` between 0 and 9, inclusive.

```
// Ask the user for a lock index in ``locks`` between 0 and 9, inclusive
std::cout << "Lock number? ";
int lock_index = 0;
std::cin >> lock_index;
```

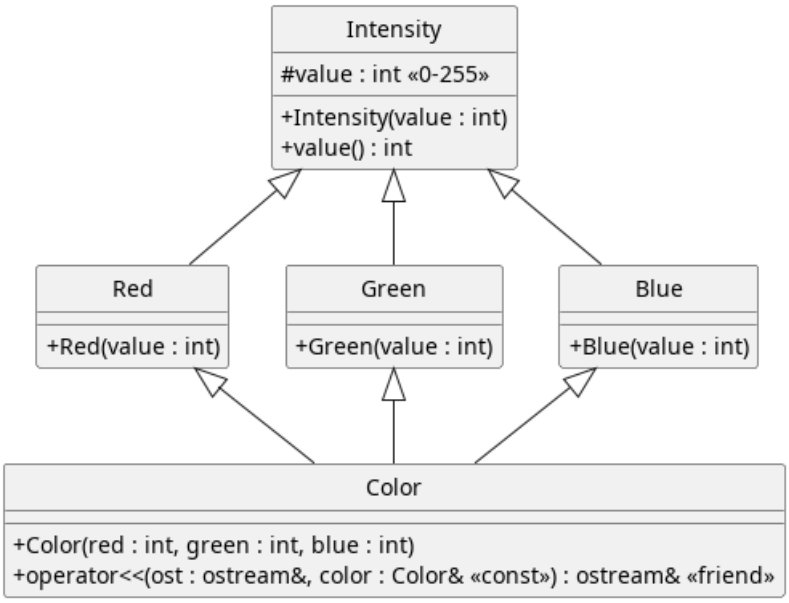


`f.` {10 points} In file keyring.cpp, write the main function.

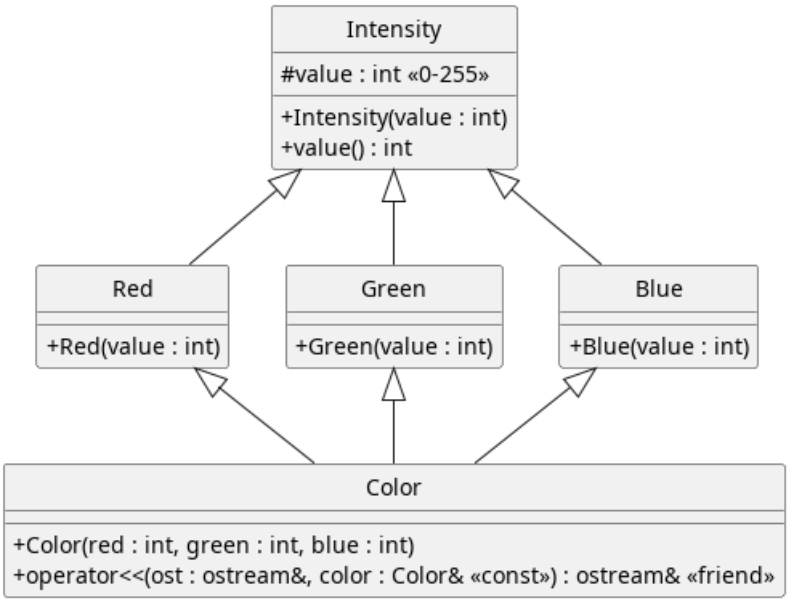
- Iterate over `keys`, trying each `Key` object in the lock object's `open` method. If the key doesn't fit, `open` will throw a `Misfit` exception, which should be ignored. But if no exception is thrown, the key fits and the lock is now open - print the index of that key that unlocked the selected lock.

```
    // Iterate over ``keys``, trying each ``Key`` object
    //   in the lock object's ``open`` method
    // The "lock" variable is NOT required
    Lock& lock = locks[lock_index];
    for(int i=0; i<NUM_LOCKS; ++i) {
        try {
            lock.open(keys[i]);
            // But if no exception is thrown, the key fits
            // and the lock is now open - print the index
            // of that key that unlocked the selected lock
            std::cout << "Lock " << lock_index
                      << " is opened by key " << i << std::endl;
            break;
        }
        // If the key doesn't fit, ``open`` will throw a ``Misfit`` exception,
        // which should be ignored.
        catch(Misfit e) {
        }
    }
}
```

3. (inheritance, multiple inheritance, iomanip, <<) Consider the following class diagram, which takes a *creative* approach to defining an RGB (red:green:blue) color definition.



3. (inheritance, multiple inheritance, iomanip, <<) Consider the following class diagram, which takes a *creative* approach to defining an RGB (red:green:blue) color definition.

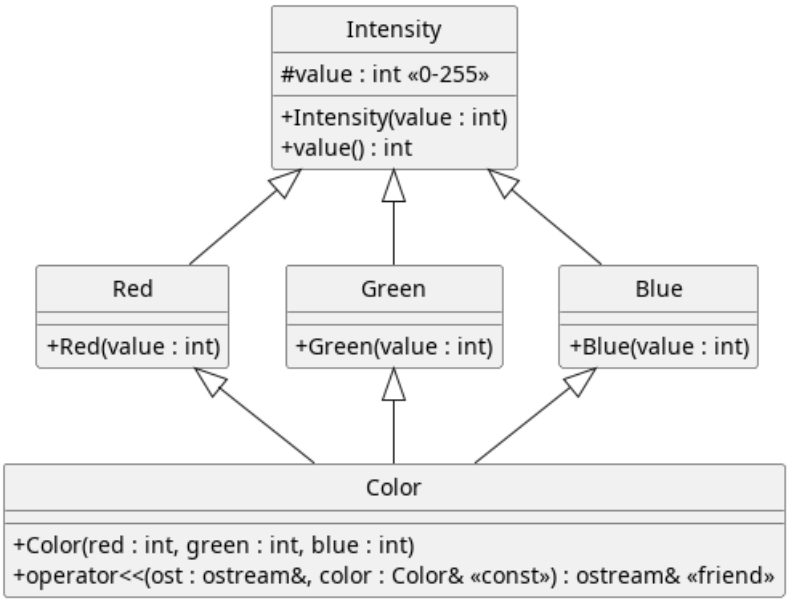


a. {4 points} In file color.h, write the declaration for class Color. (Do not write a guard or includes.) Notice the multiple inheritance in the UML diagram. You need not write the friend declaration; it is

```
friend std::ostream& operator<<(std::ostream& ost, const Color& color);
```

```
class Color : public Red, public Green, public Blue {
  public:
    Color(int red, int green, int blue);
};
```

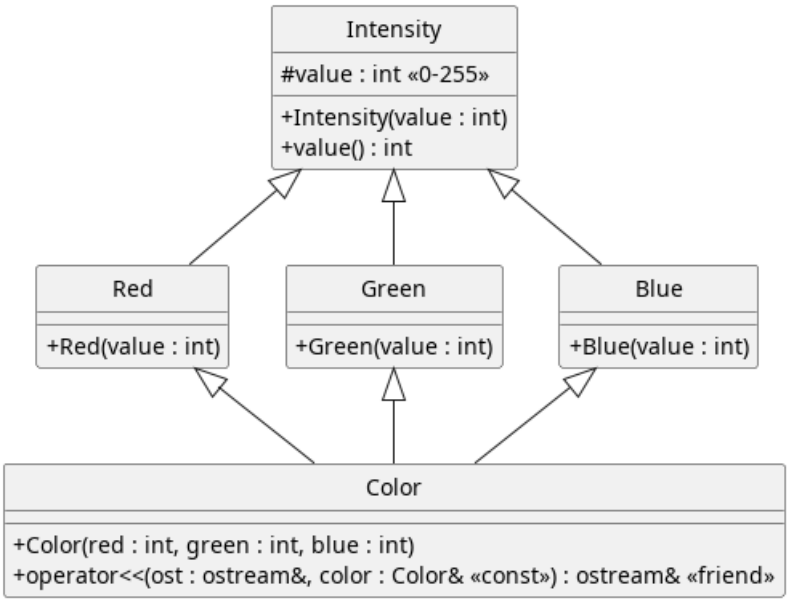
3. (inheritance, multiple inheritance, iomanip, <<) Consider the following class diagram, which takes a *creative* approach to defining an RGB (red:green:blue) color definition.



b. {4 points} In file color.cpp, write the constructor. Chain to just the immediate superclass constructors, not `Intensity` which the superclass constructors will handle individually.

```
// Curly braces in init list are OK
// No deduction if Intensity is also constructed despite requirement
Color::Color(int red, int green, int blue)
    : Red(red), Green(green), Blue(blue) { }
```

3. (inheritance, multiple inheritance, iomanip, <<) Consider the following class diagram, which takes a *creative* approach to defining an RGB (red:green:blue) color definition.



c. {3 points} In file color.cpp, write the streaming out operator for class `Color`. Using I/O manipulators, set output to hexadecimal with the fill character as '0'. Then stream out the `Red` value as 2 digits, a colon, the `Green` value as 2 digits, a colon, and the `Blue` value as 2 digits. Output should therefore look something like `ff:7f:00`.

```
std::ostream& operator<<(std::ostream& ost, const Color& color) {
    return ost << std::hex << std::setfill('0')
               << std::setw(2) << color.Red  ::value() << ":"
               << std::setw(2) << color.Green::value() << ":"
               << std::setw(2) << color.Blue ::value();
}
```

Bonus

Bonus 1: {+2 points} If an `int` can be stored on the heap, write a line of **C++** code that does that, followed by a line of code that frees its memory from the heap. If it cannot, explain why *in a single brief sentence*.

Bonus

Bonus 1: {+2 points} If an `int` can be stored on the heap, write a line of **C++** code that does that, followed by a line of code that frees its memory from the heap. If it cannot, explain why *in a single brief sentence*.

```
// It can.

int* ip = new int; // Allocate an int on the heap
delete ip;         // Free the int from the heap
```

Bonus 2: {+2 points} In one *concise* sentence, explain what a `constexpr` is.

Bonus 2: {+2 points} In one *concise* sentence, explain what a `constexpr` is.

```
// A C++ ``constexpr`` is a keyword that indicates a value, function, or
// object can be evaluated at compile time so that the result can be loaded
// with the program directly, saving initialization time and memory.
```