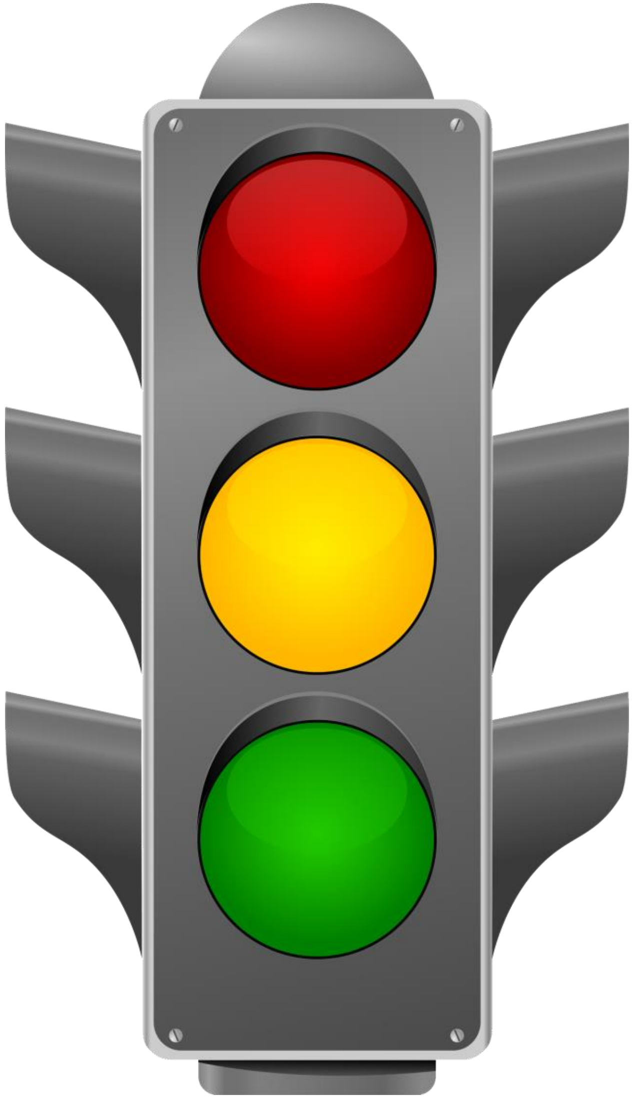
CS4414 Recitation 4
All about classes

09/17/2021

Sagar Jha

Define your own types with classes

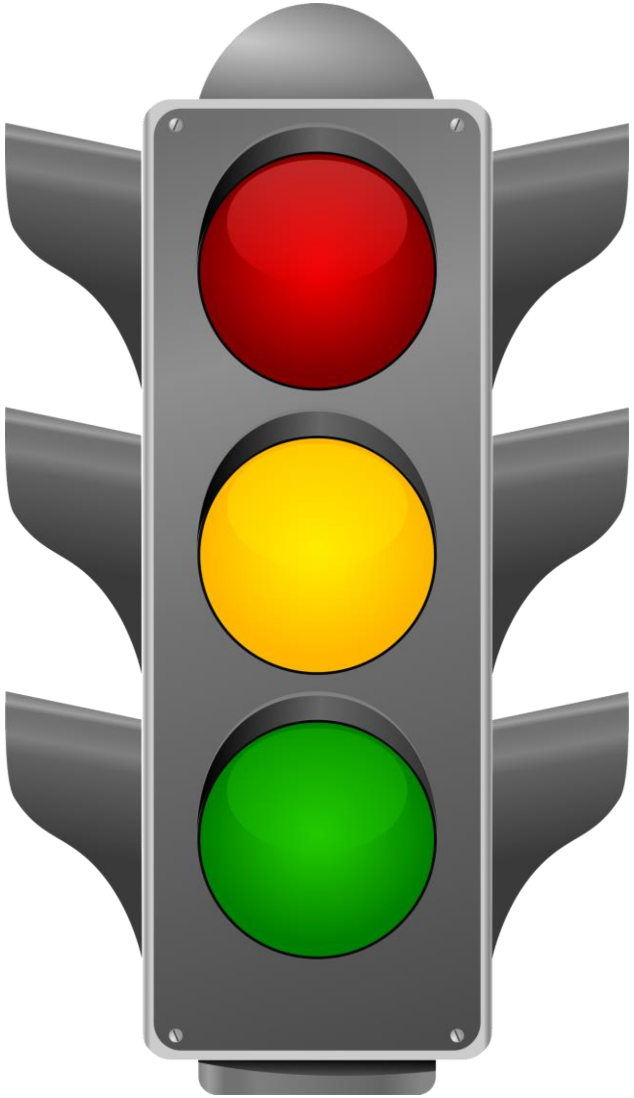
- Let’s say we wanted to write a class TrafficLight



Define your own types with classes

- Let's say we wanted to write a class TrafficLight

What is its state?	What does it do?
• color • cycle length	• displays color • changes color



Defining TrafficLight class

- *Define* the class in the header (.hpp) file
- class TrafficLight {

```
  public:
      Color getColor();
      void nextColor();
  private:
      Color color; // enum
      int length;
};
```

- A class can contain objects of other classes. For e.g., **TrafficController** will contain objects of type **TrafficLight**

Implementing the class in the source (.cpp) file

- #include “TrafficLight.hpp”
- void TrafficLight::getColor() {

  return color;

  }
- <rest of the functions>

Why separate the class definition and the implementation?

- It is standard practice!
- Improves compilation time
- Everything in the header file is compiled each time that header file is included in a translation unit (.cpp)
- Separating the implementation means that the implementation is compiled only once
- Then it is linked in the linking phase

Similar process for global functions

- Include declaration in the header file

```
void read_file (std::string& filename);
```

- Implement the function in the associated source file

```
void read_file (std::string& filename) {
    std::ifstream fin(filename);
    while (true) {
      std::string word;
      fin >> word;
      if(fin.fail()) {
        break;
      }
      // do something with word
    }
}
```

C++'s **one definition rule** (ODR)

- Each definition (for a function, variable, class etc.) must appear only once in each translation unit
- What about class definitions in the header file?
- Example – **vector.hpp** defines the class **std::vector<T>**. TrafficLight.hpp and TrafficController.hpp may both include this file
- Include **#pragma once** at the top of each header file you create

A note about compilation

- If not using Makefiles, run “g++ -o exec_name main.cpp rest.cpp …”
- Include all the cpp files in the g++ command
- Leave out the header files as they are included in the cpp files
- Only one program should contain the main function (in the above, main.cpp)

Using classes

- Once a class is defined, you can define instances (called objects)
- All objects have their own state, but share the class functions
- E.g., std::string str; // creates an empty string, “”

Using classes

- Once a class is defined, you can define instances (called objects)
- All objects have their own state, but share the class functions
- E.g., std::string str; // creates an empty string, “”
- Unlike Java, class objects are **NOT** null references in C++!
- This means that when you create an object, all of its internal fields must be initialized. When the object goes out of scope, it is destroyed (*deconstructed*).
- Each class has at least one constructor and one destructor

Default initialization in C++

- class myClass {

```
class myClass {
        int x;
        std::string str;
        std::vector<int> vec;
};
```

- This class does not **explicitly** define a constructor function
- In such cases, the compiler provides a default constructor
- The default constructor **default initializes** the fields

More about constructors

- A constructor has the same name as the class and no return type. It can have as many arguments as needed (just like a regular function)
- You can write as many constructors as you need
- E.g.,
  - **myClass();**
  - **myClass(int x, std::string str, std::vector<int> vec);**
  - **myClass(someOtherClass other) and so on**

More about constructors

- Special constructors
  - Default constructor – takes no arguments
  - Copy constructor – **myClass(const myClass& other);**
  - Move constructor – **myClass(myClass&& other);**
- The compiler provides a default constructor (public) when no constructors are defined
- It also provides a default copy and a default move constructor unless the user defines them

Different ways of creating a vector

- std::vector<int> numbers; // default constructor
- std::vector<int> numbers(5); // notice the parentheses, creates a vector of size 5, all 0s
- std::vector<int> numbers(5, 100); // all elements initialized to 100

Different ways of creating a vector

- std::vector<int> numbers; // default constructor
- std::vector<int> numbers(5); // notice the parentheses, creates a vector of size 5, all 0s
- std::vector<int> numbers(5, 100); // all elements initialized to 100
- std::vector<int> one_to_ten = {1, 2, 3, 4, 5, 6, 7,. 8, 9, 10}; // uses initializer list
- std::vector<int> numbers(one_to_ten); // one_to_ten is of type std::vector<int>, invokes the copy constructor
- How to find out about these constructors and other vector functions? Read the C++ reference!

Even more about constructors

- Using the keywords **default** and **delete**, you can enable or disable a constructor
- What if you want to disable the copy constructor? For e.g., you want unique ownership of a resource and don’t want it duplicated.
- What if you write a custom constructor that takes some arguments, but still want to keep a default constructor?
- myClass(const myClass& other) = delete;
- myClass() = default;

Exercise: Find the error!

Exercise: Find the error!

- class myClass {

public:

    myClass(int x) {}

private:

    int myInt;

};

- std::vector<myClass> myObjects(4);

Exercise: Find the error!

- class myClass {

```
public:
    myClass(int x);
private:
    int myInt;
};
```

- std::vector<myClass> myObjects(4);

Solution: The vector cannot default-construct its constituent objects!

How to rectify the error?

- push_back constructed elements

```
std::vector<myClass> myObjects; // size 0
myClass obj1(5);
myClass obj2(7);
myObjects.push_back(obj1);
myObjects.push_back(obj2);
```

- push_back will invoke the copy constructor to copy the object into the vector

Constructor initializer list

- Problem: How to construct constituent elements of a class in the constructor?
- E.g.,
  - Suppose we have **Student(std::string name);**, constructor for Student
  - Next, we have CS4414Group that contains three Student objects **A**, **B**, and **C**
  - How can we construct the Student objects, part of a group, in the constructor of CS4414Group?

Constructor initializer list

- Unlike Java, you cannot construct data members in the body of the constructor. In Java, you would do something like,

```
CS4414Group::CS4414Group() {
        this->A("Ken");
        this->B("Sagar");
        this->C("Alicia");
}
```

- But in C++, objects cannot be null. Member objects must be constructed when the enclosing class object is constructed.

Constructor initializer list

- After the signature of the constructor and before the body, include a constructor initializer list
- CS4414Group::CS4414Group(std::string& name1, std::string& name2, std::string& name3) : A(name1),

```
                                          B(name2),
                                          C(name3) {
        // the body of the constructor
}
```

- comma-separated list of the type class_member(args…)

Public and private members

- public members of a class can be accessed anywhere
- For e.g., in main function, if I create an object of type **someInt** containing a public integer x, I can do something like

```
std::cout << someIntObj.x << std::endl;
```

- Private members can only be accessed inside class’s member functions
- E.g., if x was private, the above use-case would fail. But inside someInt(const someInt& other), I can access both **this->x** and **other.x**
- But in another class, notSomeInt(someInt intObj), I cannot access intObj.x either

Question: Why should we prefer to define the data members (state variables) as private?

Question: Why should we prefer to define the data members (state variables) as private?

Most important reason:

If you “expose” a data member outside the class, you will have no control over how that member will be used.

Question: Why should we prefer to define the data members (state variables) as private?

Most important reason:

If you “expose” a data member outside the class, you will have no control over how that member will be used.

- Think back to the TrafficLight class. Its logic should control the color change from red -> green -> yellow -> red. If color was public, someone outside could break this logical progression.
- Why does it matter?
  - In a big software company, your team members will use classes you write
  - If you write a library for general-purpose use, your users will use it in all sorts of funky ways

Recommended access modifier rules

- Private for data members, except maybe const members
- Public for member functions that are needed by other parts of the code. If there are only private constructors, no objects of the class can be created outside the class.
- Private for internal helper functions

Static members of a class

- A data member that is shared by all objects of the class
- E.g., A static integer counting the total number of objects created
- Prefer static class members over globals (**Guiding principle**: Never use globals)
- In the class definition, include the definition like this:

  static int objectCount;

- Static members cannot be initialized in constructors (obviously because they don’t exist per class object)
- Initialize them in the .cpp file like this:

  int myClass::objectCount = 0; // no objects created at this point

Putting it all together: Workflow examples

- You maintain a collection of all future events in a priority_queue
- Each event is a pair consisting of a TrafficController and time
- You get the next event from the queue
- You call function transition on the TrafficController object

Putting it all together: Workflow examples

- You maintain a collection of all future events in a priority_queue
- Each event is a pair consisting of a TrafficController and time
- You get the next event from the queue
- You call function transition on the TrafficController object
- In transition, you call nextColor on the second TrafficLight object which changes color from Yellow to Red. Then you call nextColor on the third TrafficLight object which changes color from Red to Green.
- The transition function then returns the cycle length of the third traffic light back to you
- You then insert a future even with the same controller object