Programming Techniques for Scientific Simulations I

Introduction to classes

Classes

- Are a way to create new data types
  - E.g. a vector or matrix type
- Object oriented programming
  - Instead of asking: “What are the subroutines?”
  - We ask:
    - What are the abstract entities?
    - What are the properties of these entities?
    - How can they be manipulated?
  - Then we implement these entities as classes
- (Some) key advantages:
  - High level of abstraction possible
  - Hiding of representation dependent details
  - Presentation of an abstract and simple interface
  - Encapsulation of all operations on a type within a class
    - Separation of concerns (allows easier reasoning/debugging...)

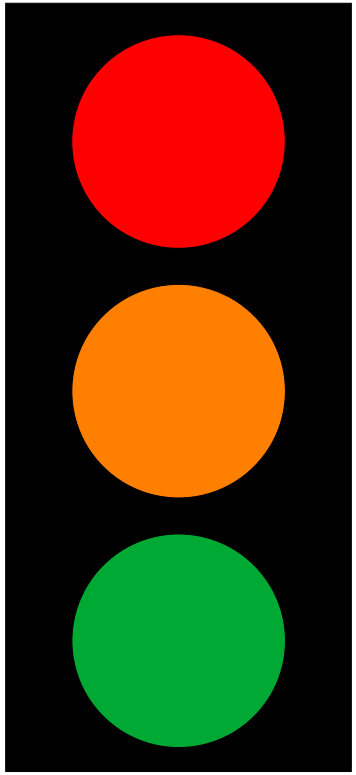
What are classes?

- Classes are collections of “members” representing one entity
- Members can be
  - Functions
  - Data
  - Types
- These members can be split into
  - `public`: accessible interface to the outside. (Should not be modified later! Why?)
  - `private`: hidden representation of the concept. (Can be changed without breaking any program using the class. Why?)
- Objects of this type can be modified only through these member functions
  - Localization of access, easier debugging

How to design classes

- Ask yourself some questions
- What are the logical entities (nouns)?
  - The classes
- What are the internal state variables?
  - The private data members
- How will it be created/initialized and destroyed?
  - The constructor (ctor) and destructor (dtor)
- What are its properties (adjectives)?
  - The public constant member functions
- How can it be manipulated (verbs)?
  - The public operators and member functions

A first class example: a traffic light



A first class example: a traffic light

- Property
  - The state of the traffic light (green, orange or red)
- Operation
  - Set the state
- Construction
  - Create a light in a default state (e.g. red)
  - Create a light in a given state
- Destruction
  - Nothing special needs to be done
- Internal representation
  - Store the state in a variable
  - Alternative: connect via a network to a real traffic light

A first class example: a traffic light

- Converting the design into a class

```
class Trafficlight {

};
```

A first class example: a traffic light

- Add a public type member

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};
};
```

Enumerated type, i.e. a set of named values

A first class example: a traffic light

- Add a private data member (variable) of that type

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};

  // access declaration
  private: // the following is hidden
    // data member
    light state_;
};
```

Private data members marked underscore.
Just a convention… Many variants possible.

A first class example: a traffic light

- Add a const member function to access the state and a non-const member function to set the state

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};



    // function member declaration
    light state() const;
    void set_state(light);

  // access declaration
  private: // the following is hidden
    // data member
    light state_;
};
```

Declares that the function does not change the state

A first class example: a traffic light

- Add a constructor (ctor)

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};
    // ctor (initialize state_ with ctor initializer list)
    Trafficlight(           ) :           {}

    // function member declaration
    light state() const;
    void set_state(light);

  // access declaration
  private: // the following is hidden
    // data member
    light state_;
};
```

(Default) ctor has the same name as the class

A first class example: a traffic light

- Constructor to construct from given light or default to red

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};
    // ctor (initialize state_ with ctor initializer list)
    Trafficlight(light l=red) : state_(l) {}

    // function member declaration
    light state() const;
    void set_state(light);

  // access declaration
  private: // the following is hidden
    // data member
    light state_;
};
```

Default argument

Initializes `state_` with `l`
So-called initializer list

A first class example: a traffic light

- And finish by adding a destructor (dtor) (called to cleanup at destruction)

```
class Trafficlight {
  // access declaration
  public: // the following is public
    // type member
    enum light {green, orange, red};
    // ctor (initialize state_ with ctor initializer list)
    Trafficlight(light l=red) : state_(l) {}
    // dtor (empty)
    ~Trafficlight() {}
```

dtor has the same name as the class prefixed by ~

```
    // function member declaration
    light state() const;
    void set_state(light);

  // access declaration
  private: // the following is hidden
    // data member
    light state_;
};
```

Data hiding and access

- The concept expressed through the class is representation-independent
- Programs using a class should thus also not depend on representation
- Access declarators
  - `public`: only representation-independent interface, accessible to all
  - `private`: representation-dependent functions and data members
  - `friend`: declarators allow related function/classes access to representation
- Note: Since all data members are representations of concepts (numbers, state, etc.) they should be hidden (private)!
- By default all members are `private`
  In a **`struct`** by default all are `public`

Member access

- Usage:

```
Trafficlight x(Trafficlight::green);
Trafficlight::light l;

l = x.state();
l = Trafficlight::green;
```

- Members accessed with

```
object_name.member_name
```

Member access operator

- Type members accessed with

```
class_name::type_member_name
```

Scope operator

as they are not bound to a specific object but common to all

```
class Trafficlight {
  public:
   enum light {green, orange, red};
    Trafficlight(light l=red) : state_(l) {}
   ~Trafficlight() {}
   light state() const;
   void set_state(light);
 private:
    light state_;
};
```