SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220202<A|D>

Replace <A|D> with this section's letter

Object Oriented Programming Basics

—

CS 2124: Object Oriented Programming
Darryl Reeves, Ph.D.

Agenda

- Classes
- Constructors
- `const` methods
- Encapsulation
- In-class problem

C++ classes

`class` format

name for `class`
(traditionally capitalized)

open curly brace

keyword indicates
`class` being defined

```
class ClassName {

    /* class details */

};
```

lots can go inside class body

close curly brace

semicolon terminates
definition (very important!)

A minimal `class`

```
class Simplest {};

int main() {
    Simplest sim;
}
```

full class definition

-- code will compile

`sim` is a variable with type `Simplest`

`sim` is an **object** created using a *default* constructor

`class` format

name for `class` (traditionally capitalized)

open curly brace

keyword indicates `class` being defined

```
class ClassName {
    public:
        /* public interface */
};
```

any members defined in public interface accessible without restriction

close curly brace

semicolon terminates definition (very important!)

A (slightly) more useful `class`

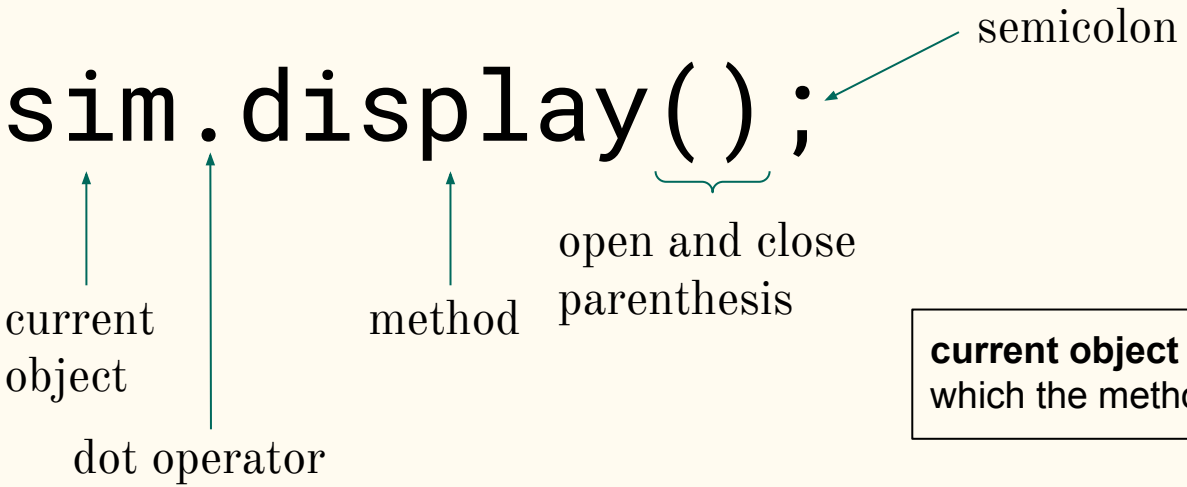
member function (**method**)

```
class Simple {
 public:
    void display() {
        cout << "Displaying a Simple object\n";
    }
};


int main() {
    Simple sim;

    sim.display();
}
```

```
Displaying a Simple object
```

Invoking an object method

```
int main() {
    Simple sim;

    sim.display();
}
```



current object - the specific object on which the method is being invoked

Differentiating class objects

```
class Vorlon {
 public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    string my_name;
};

int main() {
   Vorlon v1;

   v1.my_name = "Kosh";
   v1.display();
}
```

`my_name` has no protection from modification

```
Displaying a Vorlon named Kosh
```

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220202<A|D>

Replace <A|D> with this section's letter

What about `my_name` allows it's value to be modified without restrictions?

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    string my_name;
};

int main() {
   Vorlon v1;

   v1.my_name = "Kosh";
   v1.display();
}
```

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    string my_name;
};

int main() {
    Vorlon v1;

    v1.my_name = "Kosh";
    v1.display();
}
```

`my_name` has no protection from modification

```
Displaying a Vorlon named Kosh
```

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    string my_name;
};
```

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};

int main() {
    Vorlon v1;

    v1.my_name = "Kosh"; // Compilation ERROR!
    v1.display();
}
```

can't access private member variable my_name

`class` format

name for `class` (traditionally capitalized)

open curly brace

keyword indicates `class` being defined

```
class ClassName {
public:
    /* public interface */
private:
    /* private interface */
};
```

close curly brace

semicolon terminates definition (very important!)

private members have restricted access

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;

};

int main() {
    Vorlon v1;

    v1.my_name = "Kosh"; // Compilation ERROR!
    v1.display();
}
```

one solution: add a public method to set `Vorlon` name

can't access private member variable my_name

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    void set_name(string name) { my_name = name; }
private:
    string my_name;
};

int main() {
    Vorlon v1;

    v1.my_name = "Kosh"; // Compilation ERROR!
    v1.display();
}
```

can't access private member variable my_name

Differentiating class objects

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
    void set_name(string name) { my_name = name; }
private:
    string my_name;
};

int main() {
    Vorlon v1;

    v1.set_name("Kosh");
    v1.display();
}
```

known as a *setter* (or *mutator*) function

a better approach is available

Constructors

Initializing member variables

- special methods for initializing objects: **constructors**

```
class Vorlon {
public:
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

Initializing member variables

- special methods for initializing objects: **constructors**

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

will improve shortly

Note: no return type

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};


int main() {
    Vorlon v1;

    v1.my_name = "Kosh";
    v1.display();
}
```

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};


int main() {
    Vorlon v1;

    v1.my_name = "Kosh";
    v1.display();
}
```

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

```
int main() {
    Vorlon v1("Kosh");

    v1.display();
}
```

no need to directly access private member

Assignment vs. initialization

1

```
string cat;
cat = "Felix";
```

2

```
string cat = "Felix";
```

Which approach to creating a string with the value "Felix" is preferrable?

1

```
string cat;
cat = "Felix";
```

2

```
string cat = "Felix";
```

Assignment vs. initialization

1

```
string cat;
cat = "Felix";
```

Two steps:

1) Create an empty string
2) Assign value "Felix"

2 steps waste resources

2

```
string cat = "Felix";
```

One step: create a string with the value "Felix"

Assignment vs. initialization

1

```
string cat;
cat = "Felix";
```

Two steps:

1) Create an empty string
2) Assign value "Felix"

2 steps waste resources

2

```
//string cat = "Felix";
string cat("Felix");
```

One step: create a string with the value "Felix"

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

Two steps:

1) Create an empty string
2) Assign value

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    Vorlon(const string& a_name) : my_name(a_name) {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

One step: create a string with initial value `a_name`

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

initialization list

constructor body can be empty

`class` format

colon (`:`) indicates
initialization list to follow


```
class ClassName {
public:
    ClassName(param_type param_name, ...): initialization_list {
        /* constructor body */
    }

};
```

`class` format

colon (`:`) indicates
initialization list to follow


```
class ClassName {
public:
    ClassName(param_type param_name, ...): initialization_list {
        /* constructor body */
    }

private:
    m_type1 m_name1;
    m_type2 m_name2;
    ...
};
```

`class` format

colon (`:`) indicates
initialization list to follow


```
class ClassName {
public:
    ClassName(p_type1 p_name1, p_type2 p_name2, ...): initialization_list {
        /* constructor body */
    }

private:
    m_type1 m_name1;
    m_type2 m_name2;
    ...
};
```

`class` format

colon (`:`) indicates *initialization list* to follow

multiple members can be initialized

```
class ClassName {
public:
    ClassName(p_type1 p_name1, p_type2 p_name2, ...): m_name1(p_name1), m_name2(p_name2),... {
        /* constructor body */
    }

private:
    m_type1 m_name1;
    m_type2 m_name2;
    ...
};
```

each initialization separated by a comma (`,`)

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

speed improvement over

What if my_name should be immutable?

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    string my_name;
};
```

speed improvement over

What if my_name should be immutable?

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) {my_name = a_name;}    // Compile error!
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

What if my_name is immutable?

Initializing member variables

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}    // No compile error!
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

const member variable ensures that once initialized, member is not modifiable

The default constructor

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

```
int main() {
    Vorlon v1; // Compile error!

}
```

no compiler error when class lacks constructor definition

- *default* constructor (automatically) provided when no constructor defined
- *default* constructor **must** be defined if class contains *any* constructor definition

The default constructor

- special constructor that takes no arguments

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};


int main() {
    Vorlon v1;  // No compile error!

}
```

default constructor

The default constructor

- special constructor that takes no arguments
- initializes all member variables of a type that is itself a `class`
  - excludes `C++` built in types (`int`, `char`, `double`, etc)
  - excludes *pointer* types

pointers coming soon

What value would the member variable `my_name` have when declaring a `Vorlon` in the manner displayed below?

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};

int main() {
    Vorlon v1;

}
```

The default constructor

- special constructor that takes no arguments
- initializes all member variables with type that is a `class`
  - excludes `C++` built in types (int, char, double, etc)
  - excludes *pointer* types
- member variables without explicit initialization use default constructor

`const` methods

Passing an object to a function (review)

- pass-by-value
  - parameter value is a copy of object

wastes resources

- pass-by-reference
  - parameter value is the object

sometimes useful but requires care

- pass-by-constant-reference
  - parameter value is the (immutable) object

safe and efficient

Implications of pass-by-constant-reference

```
void simple_function(const Vorlon& fred) {
     fred.display(); // Compile error!
}
```

simple_function has declared not to modify fred

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

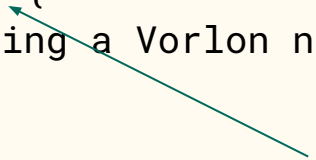
display method makes no promise to leave object unchanged

`const`: a non-modification guarantee

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

`const`: a non-modification guarantee

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() const {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};

void simple_function(const Vorlon& fred) {
     fred.display(); // Compile error!
}
```



indicates that function will not modify current object

- `const` keyword located *after* parameter list
- `const` keyword located *before* function body

Important: mark methods as `const` when no current object members modified

Encapsulation

Encapsulation

```
class Vorlon {
public:
    Vorlon(const string& a_name) : my_name(a_name) {}
    Vorlon() {}
    void display() const {
        cout << "Displaying a Vorlon named " << my_name << endl;
    }
private:
    const string my_name;
};
```

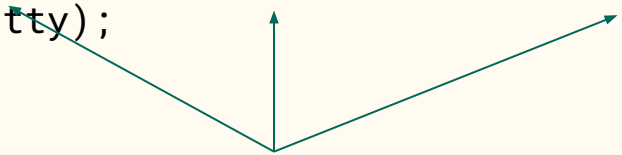
methods

data

encapsulation - bundling of data and methods that operate on the data into a single structure preventing unwanted or unintended modification

Consequences of encapsulation

```
struct Cat {
   string color;
   string name;
   double weight;
};

void fill_cat_vector(ifstream& in_fs, vector<Cat>& in_vec) {
    Cat kitty;

    while (in_fs >> kitty.name >> kitty.color >> kitty.weight) {
        in_vec.push_back(kitty);
    }

}
```



direct access to Cat member variables

Whiskers brown 8
Felix grey 6.3
Garfield orange 10.1

Consequences of encapsulation

```
class Cat {
public:
   Cat(const string& the_name, const string& the_color, double the_weight)
       : name(the_name), weight(the_weight), color(the_color) {}

   void display() const {
       cout << "Displaying a Cat named" << name << " with color ";
       cout << color << " and weight " << weight << endl;
   }

private:
    string name;
    string color;
    double weight;

};

void fill_cat_vector(ifstream& in_fs, vector<Cat>& in_vec) {
     Cat kitty; // compilation error!

     while (in_fs >> kitty.name >> kitty.color >> kitty.weight) { // compilation error!
        in_vec.push_back(kitty);
     }
}
```

Why does the declaration of `kitty` result in a compilation error?

```
class Cat {
public:
   Cat(const string& the_name, const string& the_color, double the_weight)
       : name(the_name), weight(the_weight), color(the_color) {}

   void display() const {
       cout << "Displaying a Cat named" << name << " with color ";
       cout << color << " and weight " << weight << endl;
   }

private:
    string name;
    string color;
    double weight;
};

void fill_cat_vector(ifstream& in_fs, vector<Cat>& in_vec) {
     Cat kitty; // compilation error!

     while (in_fs >> kitty.name >> kitty.color >> kitty.weight) { // compilation error!
        in_vec.push_back(kitty);
     }
}
```

Consequences of encapsulation

```
void fill_cat_vector2 (ifstream& ifs, vector<Cat>& vc) {
    string name;    // used to read in the name
    string color;   // used to read in the color
    double weight;  // used to read in the weight

    while (ifs >> name >> color >> weight) {
       Cat a_cat(name, color, weight);
       vc.push_back(a_cat);
    }
}
```

cat object defined inside loop

Whiskers brown 8
Felix grey 6.3
Garfield orange 10.1

Consequences of encapsulation

```
class Cat {
public:
   Cat(const string& the_name, const string& the_color, double the_weight)
       : name(the_name), weight(the_weight), color(the_color) {}

   void display() const {
       cout << "Displaying a Cat named" << name << " with color ";
       cout << color << " and weight " << weight << endl;
   }

private:
    string name;
    string color;
    double weight;
};


void display_cat_vector(const vector<Cat>& cat_vec) {
     for (size_t i = 0; i < cat_vec.size(); ++i) {
          cat_vec[i].display();
     }
}
```

vector passed as const reference

In-class problem

Representing people

- people can be many things
  - students
  - employees
  - customers
- create representation of person
  - start with `struct`
  - transition to `class`

```
struct Person {
    string name;
};


void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}
```

struct to class

person.cpp

```
struct Person {
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

struct to class

person.cpp

```
struct Person {
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

let's use a class instead!

```
% g++ -std=c++11 person.cpp -o person
% ./person
John eating
```

struct to class

```
struct Person {
    string name;
};
```

struct to class

```
_1_ Person {
    string name;
};
```

Which keyword replaces blank #1 to declare `Person` as a class?

```
_1_ Person {

    string name;

};
```

struct to class

```
class Person {
    string name;
};
```

struct to class

```
class Person {

    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

Why does the indicated line cause a compilation error?

```
class Person {

    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";   compilation error!
    make_eat(john);
}
```

struct to class

```
class Person {
---
    string name;

};
```

struct to class

```
class Person {
_2_
    string name;

};
```

Which keyword replaces blank #2 to make it clear which type of access the member variable `name` has?

```
class Person {
_2_
    string name;

};
```

struct to class

```
class Person {
private:
    string name;

};
```

don't forget the colon!

struct to class

```
class Person {
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

struct to class

```
class Person {
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

How can we modify the `Person` class to assign the name "John" to the `Person` object?

```
class Person {
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

struct to class

```
class Person {
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

struct to class

```
class Person {
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

struct to class

```
class Person {
---
    // define mutator function
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

struct to class

```
class Person {
_3_
    // define mutator function
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

Which keyword replaces blank #3 to make sure that the mutator function will be accessible from the `main()` function?

```
class Person {
_3_
    // define mutator function
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

struct to class

```
class Person {
public:
    // define mutator function
private:
    string name;

};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

struct to class

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

compilation error!

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

`struct` to `class`

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;


    ___
    make_eat(john);
}
```

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

`struct` to `class`

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    _4_
    make_eat(john);
}
```

class modification options:

1) **add a mutator function for `name`**
2) define a constructor

Which method invocation assigns the name "John" to the Person object john?

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    _4_
    make_eat(john);
}
```

struct to class

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.set_name("John");
    make_eat(john);
}
```

inefficient and unnecessary

class modification options:

1) ~~add a mutator function for `name`~~
2) define a constructor

struct to class

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    john.set_name("John");
    make_eat(john);
}
```

inefficient and unnecessary

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

struct to class

```
class Person {

public:
    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    john.set_name("John");
    make_eat(john);
}
```

TODO: set name when object created

inefficient and unnecessary

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    // define constructor

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;  TODO: set name when object created

    make_eat(john);
}
```

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    _5_

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

What name do we give to our constructor for the `Person` class?

```
class Person {
public:
    _5_

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    make_eat(john);
}
```

`struct` to `class`

```
class Person {
public:
    Person()

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name)

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;  TODO: set name when object created

    make_eat(john);
}
```

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

need to initialize `name`'s value

```
class Person {
public:
    Person(const string& the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;  TODO: set name when object created

    make_eat(john);
}
```

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

Which component of a constructor allows us to initialize member variables outside of the constructor's body?

`struct` to `class`

need to initialize `name`'s value

```
class Person {
public:
    Person(const string& the_name) ___ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name) _6_ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

Which token is used to indicate the start of the constructor's initialization list?

```
class Person {
public:
    Person(const string& the_name) _6_ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    make_eat(john);
}
```

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name) : {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;  TODO: set name when object created

    make_eat(john);
}
```

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name) : ___ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name) : _7_ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

Which code replaces blank #7 in order to properly initialize the member variable name?

```
class Person {
public:
    Person(const string& the_name) : _7_ {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;

    make_eat(john);
}
```

`struct` to `class`

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john;
    make_eat(john);
}
```

TODO: set name when object created

class modification options:

1) ~~add a mutator function for `name`~~
2) **define a constructor**

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john("John");

    make_eat(john);
}
```

class modification options:

1) ~~add a mutator function for `name`~~
2) **~~define a constructor~~**

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john("John");

    make_eat(john);
}
```

compilation error!

How do we update the code to avoid the compilation error?

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john("John");

    make_eat(john);
}
```

compilation error!

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void set_name(const string& the_name) { name = the_name; }
private:
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}

int main() {
    Person john("John");

    make_eat(john);
}
```

compilation error!

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    make_eat(john);
}
```

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    ___;
}
```

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    _7_;
}
```

Which code replaces blank #7 to instruct `john` to eat?

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    _7_;
}
```

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};

int main() {
    Person john("John");

    john.eat();
}
```

not quite done!

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() ___ { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};

int main() {
    Person john("John");

    john.eat();
}
```

not quite done!

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() _8_ { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};

int main() {
    Person john("John");

    john.eat();
}
```

not quite done!

Which **keyword** needs to be added to the `eat()` member function declaration to indicate that it does not change the current object?

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() _8_ { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    john.eat();
}
```

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() const { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    john.eat();
}
```

struct to class

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() const { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    john.eat();
}
```

struct to class

```
struct Person {
    string name;
};

void make_eat(const Person& a_person) {
    cout << a_person.name << " eating\n";
}


int main() {
    Person john;

    john.name = "John";
    make_eat(john);
}
```

struct to class

person.cpp

```
class Person {
public:
    Person(const string& the_name) : name(the_name) {}

    void eat() const { cout << name << " eating\n"; }

    void set_name(const string& the_name) { name = the_name; }

private:
    string name;
};


int main() {
    Person john("John");

    john.eat();
}
```

```
% g++ -std=c++11 person.cpp -o person
% ./person
John eating
```