SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220309<A|D>

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

Separate Compilation

—

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

Agenda

- Separate Compilation
- Namespaces
- Include Guards
- In-class problem

Separate Compilation

A typical program (so far)

```
#include <iostream>
using namespace std;
```

A typical program (so far)

```
#include <iostream>
using namespace std;

class Thing {
public:
     Thing(int val) : val(val) {}
     void set_val(int val) { this->val = val; }
     int get_val() const { return val; }
private:
     int val;
};
```

A typical program (so far)

```
#include <iostream>
using namespace std;

class Thing {
  friend ostream& operator<<(ostream& os, const Thing& a_thing) {
     os << "Thing: val = " << a_thing.val;
     return os;
   }
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
  Thing thing_one(17);
  cout << thing_one << endl;
  thing_one.set_val(42);
  cout << thing_one << endl;
  cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

A typical program (so far)

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing)
    {
        os << "Thing: val = " << a_thing.val;
       return os;
    }
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Split file to increase usability

- class definition (**Thing.h**)
  - method prototypes
  - associated function prototypes
- class implementation (**Thing.cpp**)
  - method definitions
  - associated function definitions
- main program (**test_thing.cpp**)
  - main function definition

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing)
    {
        os << "Thing: val = " << a_thing.val;
       return os;
    }
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

test_thing.cpp (cont.)

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing)
    {
        os << "Thing: val = " << a_thing.val;
       return os;
    }
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

start with operator<<

test_thing.cpp (cont.)

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing)
    {
        os << "Thing: val = " << a_thing.val;
       return os;
    }
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

start with operator<<

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}
```

friend designation remains in class

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

start with operator<<

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}
```

friend designation remains in class

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val) { this->val = val; }
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: set_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void set_val(int val) {
    this->val = val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: set_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void set_val(int val) {
    this->val = val;
}
```

compilation error

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: set_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void set_val(int val) {
    this->val = val;
}
```

compilation error -- need to qualify method

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: set_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}
```

compilation error -- need to qualify method

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: set_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}
```

~~compilation error --~~

~~need to qualify method~~

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const { return val; }
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: get_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: get_val()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

is const keyword part of prototype?

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

is const keyword part of prototype? yes!

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: Thing()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}
```

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val) : val(val) {}
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: Thing()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

initialization list part of definition

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

next up: Thing()

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

initialization list part of definition

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

all definitions outside of the class ✔

Step 1: Move definitions outside of class

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.cpp

Split file to increase usability

- class definition (**Thing.h**)
  - method prototypes
  - associated function prototypes
- class implementation (**Thing.cpp**)
  - method definitions
  - associated function definitions
- main program (**test_thing.cpp**)
  - main function definition

Step 2: Move implementation to .cpp file

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

test_thing.cpp (cont.)

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.cpp

Step 3: Move class definition to .h file

test_thing.cpp

```
#include <iostream>
using namespace std;

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

compiler asks:

What is a Thing?

compilation error

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

compilation error

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

compilation error

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Note: file name enclosed in " "

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.cpp

```
ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp compilation problems:

- What is an `ostream`?
- What is a `Thing`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.cpp

```
#include "Thing.h"


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp compilation problems:

- What is an `ostream`?
- What is a `Thing`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.cpp compilation problems:

- What is an `ostream`?
- What is a `Thing`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.cpp compilation problems:

- What is an `ostream`?
- What is a `Thing`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.cpp compilation problems:

- What is an `ostream`?
- What is a `Thing`?

problems solved!

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h compilation problem:

- What is an `ostream`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
#include <iostream>

class Thing {
    friend ostream& operator<<(ostream& os, const Thing& a_thing);
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h compilation problem:

- What is an `ostream`?

Step 4: Make symbols available to compiler

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
#include <iostream>

class Thing {
    friend std::ostream& operator<<(
        std::ostream& os, const Thing& a_thing
    );
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Thing.h compilation problem:

- What is an `ostream`?

problems solved!

Step 5: Compile and run

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
#include <iostream>

class Thing {
    friend std::ostream& operator<<(
        std::ostream& os, const Thing& a_thing
    );
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

```
% g++ -std=c++11 Thing.cpp test_thing.cpp -o test_thing.o
% ./test_thing.o
Thing: val = 17
Thing: val = 42
thing_one's val: 42
```

Namespaces

Making names unambiguous

```
int main() {
    string animal = "bat";  compilation error

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
using namespace std;

int main() {
    string animal = "bat"; compilation error

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
using namespace std;

int main() {
    string animal = "bat"; ~~compilation error~~

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
using namespace std;

int main() {
    string animal = "bat";

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
//using namespace std;

int main() {
    string animal = "bat"; compilation error

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
//using namespace std;

int main() {
    std::string animal = "bat"; ~~compilation error~~

    animal[0] += 1;

    cout << animal << endl;
}
```

Making names unambiguous

```
#include <string>
//using namespace std;

int main() {
    std::string animal = "bat";

    animal[0] += 1;

    cout << animal << endl;
}
```

- a *namespace* allows names to be shared (without ambiguity) within a program
  - types, functions, variables, etc
- another `string` type can be defined ("**String.h**")
  - `std::string` defined in `std` namespace

Creating a namespace

```
namespace CS2124 {
    // Thing class and function definitions
}
```

Creating a namespace

```
namespace CS2124 {
    // Thing class and function definitions
}
```

no colon

Creating a namespace

Thing.h

```
class Thing {
    friend std::ostream& operator<<(
        std::ostream& os, const Thing& a_thing
    );
public:
    Thing(int val);
    void set_val(int val);
    int get_val() const;
private:
    int val;
};
```

Creating a namespace

Thing.h

```
namespace CS2124{

    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };

}
```

Creating a namespace

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


ostream& operator<<(ostream& os, const Thing& a_thing) {
    os << "Thing: val = " << a_thing.val;
    return os;
}

void Thing::set_val(int val) {
    this->val = val;
}

int Thing::get_val() const {
    return val;
}

Thing::Thing(int val) : val(val) {}
```

Creating a namespace

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;

namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;


int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

compilation error

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;


int main() {
    CS2124::Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

~~compilation error~~

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;


int main() {
    CS2124::Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

compilation error

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;
using namespace CS2124;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

~~compilation error~~

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Using a namespace

test_thing.cpp

```
#include <iostream>
#include "Thing.h"
using namespace std;
using namespace CS2124;

int main() {
    Thing thing_one(17);
    cout << thing_one << endl;
    thing_one.set_val(42);
    cout << thing_one << endl;
    cout << "thing_one's val: " << thing_one.get_val() << endl;
}
```

Thing.h

```
#include <iostream>

namespace CS2124{
    class Thing {
        friend std::ostream& operator<<(
            std::ostream& os, const Thing& a_thing
        );
    public:
        Thing(int val);
        void set_val(int val);
        int get_val() const;
    private:
        int val;
    };
}
```

Thing.cpp

```
#include "Thing.h"
#include <iostream>
using namespace std;


namespace CS2124 {
    ostream& operator<<(ostream& os, const Thing& a_thing) {
        os << "Thing: val = " << a_thing.val;
       return os;
    }

    void Thing::set_val(int val) {
        this->val = val;
    }

    int Thing::get_val() const {
        return val;
    }

    Thing::Thing(int val) : val(val) {}
}
```

Include guards

Preventing multiple definitions of symbols

- attempting to define symbols more than once → compilation error
  - applies to functions, classes, variables, etc
- classes often defined in header files
- header files can be included in more than one file
  - first include of header defines class
  - subsequent includes of header define class again
- include guards (or macroguards) prevent multiple definitions

Include guards

Components:

```
1.  #ifndef <identifier_name>
2.  #define <identifier_name>
3.  #endif
```

<identifier_name> replaced by value naming what is being protected

Include guards

Components:

1. `#ifndef <identifier_name>`
   - means "if <identifier_name> is not defined"
   - when <identifier_name> defined everything until `#endif` ignored
2. `#define <identifier_name>`
3. `#endif`

Include guards

Components:

1. `#ifndef <identifier_name>`
2. `#define <identifier_name>`
   - defines what <identifier_name> represents (the symbol being protected)
3. `#endif`

Include guards

Components:

1. `#ifndef <identifier_name>`
2. `#define <identifier_name>`
3. `#endif`

- placed at the end of header file
- completes if statement started at `#ifndef`

Include guards

Components:

```
1.  #ifndef <identifier_name>
2.  #define <identifier_name>
3.  #endif
```

<identifier_name> replaced by value naming what is being protected

- *<identifier_name>* must be unique symbol
- *<identifier_name>* based on filename recommended
  - e.g. PERSON_H
- no periods allowed in *<identifier_name>*
- capital letters used for constants
  - e.g. PERSON_H vs. person_h

An include guard example

```
class Person {
    // rest of the definition of Person class here
};
```

An include guard example

```
#ifndef PERSON_H
#define PERSON_H
class Person {
    // rest of the definition of Person class here
};
#endif
```

Note: Any previous use of include guard with PERSON_H as <identifier_name> results in class definition being ignored

In-class problem

The (revised) program

```
int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
            << naveen << endl;
}
```

```
% g++ -std=c++11 test_princess_V2.cpp -o test_princess_V2.o
% ./test_princess_V2.o
Princess: Tiana; Single
Frog Prince: Naveen
Princess: Tiana; Married to Naveen
Frog Prince: Naveen
```

(Revised) class implementations

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend ostream& operator<<(ostream&,
  const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}

// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

Program structure

- **test_princess.cpp**
- **Princess.h**
- **Princess.cpp**
- **FrogPrince.h**
- **FrogPrince.cpp**

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220309<A|D>

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

In which file will the `main()` function be located?

```
int main() {
  Princess tiana("Tiana");
  cout << tiana << endl;

  FrogPrince naveen("Naveen");
  cout << naveen << endl;

  tiana.marry(naveen);
  cout << tiana << endl
           << naveen << endl;
}
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: main program

test_princess.cpp

```
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Separate compilation: main program

test_princess.cpp

```
---
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Separate compilation: main program

test_princess.cpp

```
#include "_1_"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Which file replaces blank #1 in order to allow declaration of a `Princess` object named `tiana`?

test_princess.cpp

```
#include "_1_"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: main program

test_princess.cpp

```
#include "Princess.h"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Separate compilation: main program

test_princess.cpp

```
#include "Princess.h"
---
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Separate compilation: main program

test_princess.cpp

```
#include "Princess.h"
#include "_2_"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Which file replaces blank #2 in order to allow declaration of a `FrogPrince` object named `naveen`?

test_princess.cpp

```
#include "Princess.h"
#include "_2_"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: main program

test_princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <iostream>
using namespace std;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Class implementations

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};
```

```
class FrogPrince {
friend ostream& operator<<(ostream&,
  const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}

// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

In which file will the `Princess` class definition be located?

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: Princess.h

Princess.h

```
#include <string>
#include <iostream>

class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&, const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};
```

What needs to be prepended to `string` and `ostream` to prevent compilation errors?

Princess.h

```
#include <string>
#include <iostream>

class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&, const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};
```

Separate compilation: Princess.h

Princess.h

```
#include <string>
#include <iostream>

class FrogPrince;

class Princess {
friend std::ostream& operator<<(std::ostream&, const Princess&);
public:
    Princess(const std::string& name);
    void marry(FrogPrince& fiance);

private:
    std::string name;
    FrogPrince* spouse;
};
```

Class implementations

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};
```

```
class FrogPrince {
friend ostream& operator<<(ostream&,
  const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}

// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

In which file will the `FrogPrince` class definition be located?

```
class FrogPrince {
friend ostream& operator<<(ostream&, const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: FrogPrince.h

FrogPrince.h

```
#include <string>
#include <iostream>

class FrogPrince {
friend ostream& operator<<(ostream&, const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

Separate compilation: FrogPrince.h

FrogPrince.h

```
#include <string>
#include <iostream>

class FrogPrince {
friend std::ostream& operator<<(std::ostream&, const FrogPrince&);
public:
    FrogPrince(const std::string& name);
    const std::string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    std::string name;
    Princess* spouse;
};
```

Class implementations

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend ostream& operator<<(ostream&,
  const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}
```

```
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

In which file will the `Princess` method definitions be located?

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: Princess.cpp

Princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <string>
#include <iostream>
using namespace std;

Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}
```

Class implementations

```
class FrogPrince;

class Princess {
friend ostream& operator<<(ostream&,
    const Princess&);
public:
    Princess(const string& name);
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend ostream& operator<<(ostream&,
  const FrogPrince&);
public:
    FrogPrince(const string& name);
    const string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    string name;
    Princess* spouse;
};
```

```
// Princess method definitions
Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}
```

```
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

In which file will the `FrogPrince` method definitions be located?

```
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

A. **test_princess.cpp**
B. **Princess.h**
C. **Princess.cpp**
D. **FrogPrince.h**
E. **FrogPrince.cpp**

Separate compilation: FrogPrince.cpp

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

Creating a namespace

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

ostream& operator<<(ostream& os, const FrogPrince& frog) {
    cout << "Frog: " << frog.name << endl;
    return os;
}

const string& FrogPrince::get_name() const { return name; }

void set_spouse(Princess* spouse) { this->spouse = spouse; }
```

Creating a namespace

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

___ {

    FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

    ostream& operator<<(ostream& os, const FrogPrince& frog) {
        cout << "Frog: " << frog.name << endl;
       return os;
    }

    const string& FrogPrince::get_name() const { return name; }

    void set_spouse(Princess* spouse) { this->spouse = spouse; }
}
```

Creating a namespace

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

_3_ {

    FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

    ostream& operator<<(ostream& os, const FrogPrince& frog) {
        cout << "Frog: " << frog.name << endl;
       return os;
    }

    const string& FrogPrince::get_name() const { return name; }

    void set_spouse(Princess* spouse) { this->spouse = spouse; }
}
```

What replaces blank #3 in order to add the `FrogPrince` methods to a `namespace` named `Fantasy`?

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

_3_ {

    FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

    ostream& operator<<(ostream& os, const FrogPrince& frog) {
        cout << "Frog: " << frog.name << endl;
       return os;
    }

    const string& FrogPrince::get_name() const { return name; }

    void set_spouse(Princess* spouse) { this->spouse = spouse; }
}
```

Creating a namespace

FrogPrince.cpp

```
#include "FrogPrince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {

    FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

    ostream& operator<<(ostream& os, const FrogPrince& frog) {
        cout << "Frog: " << frog.name << endl;
       return os;
    }

    const string& FrogPrince::get_name() const { return name; }

    void set_spouse(Princess* spouse) { this->spouse = spouse; }
}
```

Creating a namespace

Princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <string>
#include <iostream>
using namespace std;

Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

void Princess::marry(FrogPrince& fiance) {
    spouse = &fiance;
    fiance.set_spouse(this);
}

ostream& operator<<(ostream& os, const Princess& princess){
    os << "Princess: " << name;
    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );
    return os
}
```

Creating a namespace

Princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {

    Princess::Princess(const string& name) : name(name), spouse(nullptr) {}

    void Princess::marry(FrogPrince& fiance) {
        spouse = &fiance;
        fiance.set_spouse(this);
    }

    ostream& operator<<(ostream& os, const Princess& princess){
        os << "Princess: " << name;
        os << (princess.spouse == nullptr ?
           "; Single" : "; Married to " + princess.spouse->get_name() );
        return os
    }
}
```

Creating a namespace

FrogPrince.h

```
#include <string>
#include <iostream>

class FrogPrince {
friend std::ostream& operator<<(std::ostream&, const FrogPrince&);
public:
    FrogPrince(const std::string& name);
    const std::string& get_name() const;
    void set_spouse(Princess* spouse);

private:
    std::string name;
    Princess* spouse;
};
```

Creating a namespace

FrogPrince.h

```
#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince {
    friend std::ostream& operator<<(std::ostream&, const FrogPrince&);
    public:
        FrogPrince(const std::string& name);
        const std::string& get_name() const;
        void set_spouse(Princess* spouse);

    private:
        std::string name;
        Princess* spouse;
    };
}
```

Creating a namespace

Princess.h

```
#include <string>
#include <iostream>

class FrogPrince;

class Princess {
friend std::ostream& operator<<(std::ostream&, const Princess&);
public:
    Princess(const std::string& name);
    void marry(FrogPrince& fiance);

private:
    std::string name;
    FrogPrince* spouse;
};
```

Creating a namespace

Princess.h

```
#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince;

    class Princess {
    friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marry(FrogPrince& fiance);

    private:
        std::string name;
        FrogPrince* spouse;
    };
}
```

Creating a namespace

test_princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <iostream>
using namespace std;
---

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Creating a namespace

test_princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <iostream>
using namespace std;
_4_

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Which directive replaces blank #4 in order to utilize the symbols from the `Fantasy namespace` in the **test_princess.cpp** file?

test_princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <iostream>
using namespace std;
_4_

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Creating a namespace

test_princess.cpp

```
#include "Princess.h"
#include "FrogPrince.h"
#include <iostream>
using namespace std;
using namespace Fantasy;

int main() {
    Princess tiana("Tiana");
    cout << tiana << endl;

    FrogPrince naveen("Naveen");
    cout << naveen << endl;

    tiana.marry(naveen);
    cout << tiana << endl
         << naveen << endl;
}
```

Adding include guards

Princess.h

```
#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince;

    class Princess {
    friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marry(FrogPrince& fiance);

    private:
        std::string name;
        FrogPrince* spouse;
    };
}
```

Adding include guards

Princess.h

```
#ifndef PRINCESS_H
#define PRINCESS_H

#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince;

    class Princess {
    friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marry(FrogPrince& fiance);

    private:
        std::string name;
        FrogPrince* spouse;
    };
}

#endif
```

Adding include guards

FrogPrince.h

```
#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince {
    friend std::ostream& operator<<(std::ostream&, const FrogPrince&);
    public:
        FrogPrince(const std::string& name);
        const std::string& get_name() const;
        void set_spouse(Princess* spouse);

    private:
        std::string name;
        Princess* spouse;
    };
}
```

Adding include guards

FrogPrince.h

```
#ifndef FROGPRINCE_H
#define FROGPRINCE_H

#include <string>
#include <iostream>

namespace Fantasy {

    class FrogPrince {
    friend std::ostream& operator<<(std::ostream&, const FrogPrince&);
    public:
        FrogPrince(const std::string& name);
        const std::string& get_name() const;
        void set_spouse(Princess* spouse);

    private:
        std::string name;
        Princess* spouse;
    };
}

#endif
```

For next time

Read Professor Sterling's notes on operator overloading **before** class!!

https://cse.engineering.nyu.edu/jsterling/cs2124/LectureNotes/05.OverloadingOperators.html

(you should always be reading in preparation for class)