SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220307<A|D>

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

Cyclic Association

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

Agenda

- Cyclic association
- In-class problem

Cyclic Association

Forward class declarations

```
class A {

private:
   B x;
};
```

- `x` is private member of class A
- each `A` instance includes an instance of `B`
- total size of `A` instance dependent on size of `B` instance
- class `B` must be defined first

Forward class declarations

```
class C {

private:
   D* y;
};
```

- `y` is private member of class `C`
- each `C` instance includes a *pointer* to an instance of `D`
- all pointers are of the same size
- only requirement: class `D` is valid type

Forward class declarations

```
class D;

class C {


private:
   D* y;

};
```

compiler informed of class D's existence (forward declaration)

- `y` is private member of class C
- each `C` instance includes a *pointer* to an instance of `D`
- all pointers are of the same size
- only requirement: class `D` is valid type

Cyclic dependencies

```
class D {

private:
   int z;
};

class C {

private:
   D* y;
};
```

simple class --

full definition can be provided

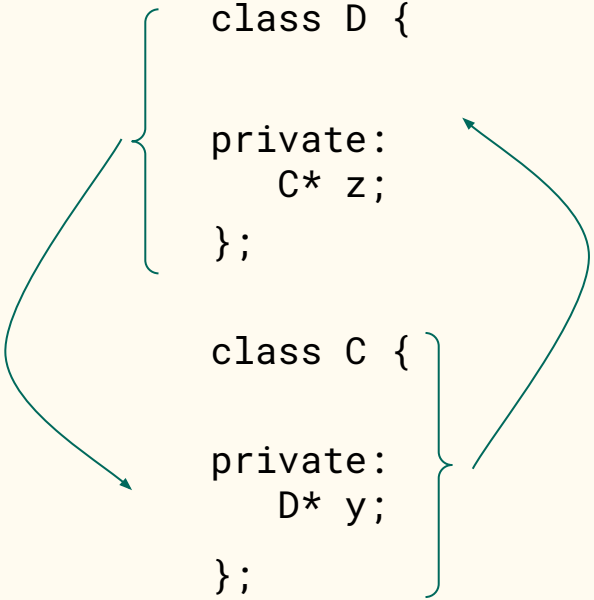
- `y` is private member of class C
- each `C` instance includes a *pointer* to an instance of `D`
- all pointers are of the same size
- only requirement: class `D` is valid type

Cyclic dependencies

```
class D {

private:
   C* z;
};

class C {

private:
   D* y;
};
```

problematic

Cyclic dependencies



Cyclic dependencies

```
class C {


private:
   D* y;
};
```

same problem

```
class D {


private:
   C* z;
};
```

Cyclic dependencies

```
class C;
```

problem solved

```
class D {

private:
   C* z;
};

class C {

private:
   D* y;
};
```

Cyclic dependencies

```
class C;

class D {

private:
   C* z;
};

class C {

private:
   D* y;
};
```

Forward declarations for classes:

- no size information
- no details on member variables
- no details on member functions
- **only** provides name of type

Cyclic dependencies

```
class C;


class D {
public:
   void d_func() { z->c_func(); }
private:
   C* z;
};
class C {
public:
   void c_func() { }
private:
   D* y;
};
```

problematic

Forward declarations for classes:

- no size information
- no details on member variables
- no details on member functions
- **only** provides name of type

Cyclic dependencies

```
class C;


class D {
public:
   void d_func() { z->c_func(); }
private:
   C* z;
};
class C {
public:
   void c_func() { }
private:
   D* y;
};
```

problematic

move definition
outside of class

Forward declarations for classes:

- no size information
- no details on member variables
- no details on member functions
- **only** provides name of type

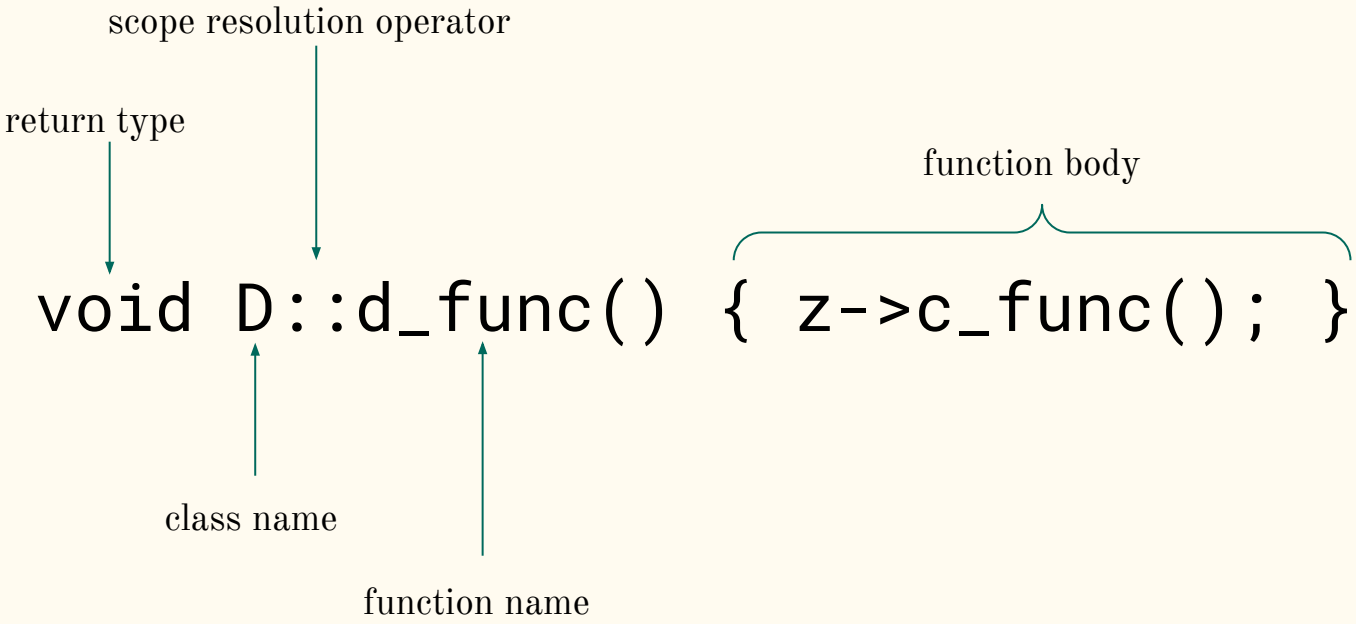
Cyclic dependencies

```
class C;

class D {
public:
   void d_func();
private:
   C* z;
};

class C {
public:
   void c_func() { }
private:
   D* y;
};

void D::d_func() { z->c_func(); }
```

member function prototype

member function definition

Defining member function outside of class



Cyclic dependencies

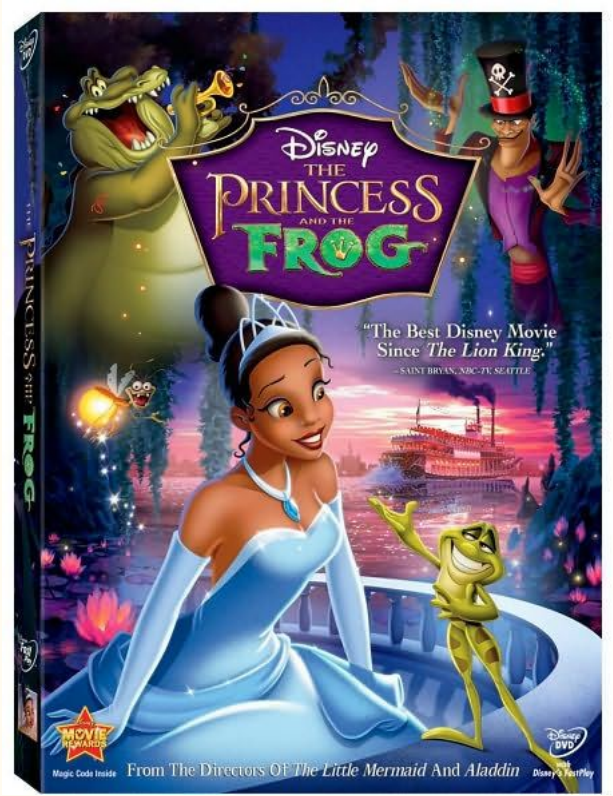
```
class C;

class D {
public:
   void d_func();
private:
   C* z;
};

class C {
public:
   void c_func() { }
private:
   D* y;
};

void D::d_func() { z->c_func(); }
```

member function prototype

member function definition

In-class problem

A Disney-inspired example



Review: Person relationships

Enabling Person objects to marry!

as long as neither already married

Person class marriage

```
class Person {
    friend ostream& operator<<(ostream& os, const Person& rhs) {
        os << "Name: " << rhs.name << ", ";

        if (rhs.spouse == nullptr) {
            os << "Single";
        } else {
            os << "Married to ";
            os << rhs.spouse->name;
        }

        return os;
    }

public:
    Person(const string& name) : name(name), spouse(nullptr) {}
    bool marry(Person& fiance) {

       if (fiance.spouse == nullptr && this->spouse == nullptr) {
          this->spouse = &fiance;
          fiance.spouse = this;

          return true;
       }
       return false;
    }
private:
    string name;
    Person* spouse;
};
```

```
int main() {
    Person john("John");
    Person mary("Mary");

    john.marry(mary);
    cout << john << '\n' << mary << '\n';


    Person bill("Bill");

    john.marry(bill);
    cout << '\n' << john << '\n';
    cout << bill << '\n' << mary << '\n';
}
```

```
% g++ -std=c++11 person.cpp -o person.o
% ./person.o
Name: John, Married to Mary
Name: Mary, Married to John

Name: John, Married to Mary
Name: Bill, Single
Name: Mary, Married to John
```

Marriage between different classes

Name: Tiana
Class: Princess

How can two instances of different classes get married?

Name: Naveen
Class: FrogPrince

The program

```
int main() {
    Princess tiana("Tiana");
    tiana.display();
}
```

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
```

The program

```
int main() {
    Princess tiana("Tiana");
    tiana.display();

    FrogPrince naveen("Naveen");
    naveen.display();
}
```

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
```

The program

```
int main() {
    Princess tiana("Tiana");
    tiana.display();

    FrogPrince naveen("Naveen");
    naveen.display();

    tiana.marry(naveen);
    tiana.display();
}
```

Requirements:

- `Princess` class
  - `name` member
  - `spouse` member
  - `display()` method
  - `marry()` method
- `FrogPrince` class
  - `name` member
  - `spouse` member
  - `display()` method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

The `Princess` class

```
class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);
private:
    string name;
    FrogPrince* spouse;
};
```

compilation error

compilation error

The `Princess` class

```
---

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);     compilation error

private:
    string name;
    FrogPrince* spouse;  compilation error

};
```

The `Princess` class

```
_1_

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);     compilation error

private:
    string name;
    FrogPrince* spouse;  compilation error

};
```

TurningPoint

SRS Setup
Login: student.turningtechnologies.com
Session ID: 20220307<A|D>

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

Which statement replaces blank #1 to eliminate the compilation errors?

```
_1_

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};
```

compilation error

compilation error

The `Princess` class

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};
```

compilation error

compilation error

The `Princess` class

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};
```

The `FrogPrince` class

```
class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

compilation error??

Would a compilation error result from the class definitions as provided?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

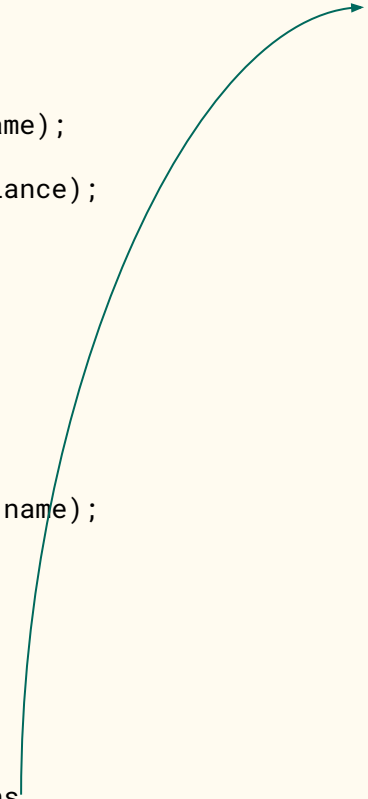
compilation error??

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};

// Princess method definitions
```



Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

```
// Princess method definitions
_2_Princess(const string& name) : name(name), spouse(nullptr) {}
```

What needs to be prepended to the constructor name (replacing blank #2) to enable the constructor to be defined outside of the `Princess` class?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

```
// Princess method definitions
_2_Princess(const string& name) : name(name), spouse(nullptr) {}
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

private:
    string name;
    Princess* spouse;

};
```

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

void Princess::marry(FrogPrince& fiance) {


}
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

Why does the second line of the `Princess::marry()` function body result in a compilation error?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

private member of `FrogPrince` class

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
___ class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

private member of `FrogPrince` class

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
_3_ class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

private member of `FrogPrince` class

Which keyword replaces blank #3 to give the `Princess` class access to private members of the `FrogPrince` class?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
_3_ class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

private member of `FrogPrince` class

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}
```

compilation error

private member of `FrogPrince` class

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << ___ << endl;
    }
}
```

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;

};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    ___ get_name() ___;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    ___ get_name() _4_;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Which keyword replaces blank #4 to guarantee that the `get_name()` method will not modify the current object?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    ___ get_name() _4_;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    ___ get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    _5_ get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Which return type replaces blank #5?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    _5_ get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

Princess::display()

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->name << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << ___ << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << _6_ << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Which expression replaces blank #6 to invoke a `get_name()` method on the `FrogPrince` object pointed to by `spouse`?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << _6_ << endl;
    }
}
```

fine -- but good motivation for accessor method

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}
```

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

`Princess::display()`

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

// FrogPrince method definitions
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

// FrogPrince method definitions
___
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

// FrogPrince method definitions
_7_
```

Which name (replacing blank #7) is used to implement a `FrogPrince` constructor outside of the `FrogPrince` class?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

// FrogPrince method definitions
_7_
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

// FrogPrince method definitions
FrogPrince::FrogPrince() {}
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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

too much access??

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

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

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

What is another option that we have for modifying the `spouse` member variable of the `FrogPrince` without using `friend` access?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
friend class Princess;
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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

too much access??

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

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

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method

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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method
    void set_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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method
    void set_spouse(___ 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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method
    void set_spouse(_8_ 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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Which type replaces blank #8 for the `spouse` parameter of the `set_spouse()` method?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method
    void set_spouse(_8_ 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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    const string& get_name() const;
    // spouse mutator method
    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.spouse = this;
}
```

compilation error

```
void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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.spouse = this;
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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;
    // set spouse
    ___
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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;
    // set spouse
    ___
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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;
    // set spouse
    _9_
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Which method invocation replaces blank #9 to set `fiance`'s spouse to be the current `Princess` object?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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;
    // set spouse
    _9_
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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;
    // set spouse
    fiance.set_spouse(this);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

compilation error

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}
```

~~compilation error~~

```
void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

// implement set_spouse
void FrogPrince::set_spouse(Princess* spouse) { ___ }
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

// implement set_spouse
void FrogPrince::set_spouse(Princess* spouse) { _10_ }
```

Which assignment statement replaces blank #10 to assign the `Princess` object pointed to by `spouse` as the spouse of the current `FrogPrince` object?

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

// implement set_spouse
void FrogPrince::set_spouse(Princess* spouse) { _10_ }
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}
// FrogPrince method definitions
FrogPrince::FrogPrince(const string& name) : name(name), spouse(nullptr) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

// implement set_spouse
void FrogPrince::set_spouse(Princess* spouse) { this->spouse = spouse; }
```

Class implementations

```
class FrogPrince;

class Princess {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

The program

```
int main() {
    Princess tiana("Tiana");
    tiana.display();

    FrogPrince naveen("Naveen");
    naveen.display();

    tiana.marry(naveen);
    tiana.display();
}
```

Requirements:

- `Princess` class
  - `name` member
  - `spouse` member
  - `display()` method
  - `marry()` method
- `FrogPrince` class
  - `name` member
  - `spouse` member
  - `display()` method

All done!

```
% g++ -std=c++11 test_princess.cpp -o test_princess.o
% ./test_princess.o
Princess: Tiana; spouse: none
Frog: Naveen
Princess: Tiana; spouse: Naveen
```

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 {
public:
    Princess(const string& name);
    void display() const;
    void marry(FrogPrince& fiance);

private:
    string name;
    FrogPrince* spouse;
};

class FrogPrince {
public:
    FrogPrince(const string& name);
    void display() const;
    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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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);
}

void Princess::display() const {
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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){
    cout << "Princess: " << name << "; spouse: ";
    if (spouse == nullptr) {
        cout << "none\n";
    } else {
        cout << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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 << "; spouse: ";
    if (spouse == nullptr) {
        os << "none\n";
    } else {
        os << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << princess.name;
    if (spouse == nullptr) {
        os << "none\n";
    } else {
        os << spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << princess.name;
    if (princess.spouse == nullptr) {
        os << "none\n";
    } else {
        os << princess.spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << princess.name;
    if (princess.spouse == nullptr) {
        os << "; Single";
    } else {
        os << princess.spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << princess.name;
    if (princess.spouse == nullptr) {
        os << "; Single";
    } else {
        os << "; Married to " << princess.spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << princess.name;
    if (princess.spouse == nullptr) {
        os << "; Single";
    } else {
        os << "; Married to " << princess.spouse->get_name() << endl;
    }
}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

space needed!!

(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: " << princess.name;

    os << (princess.spouse == nullptr ?
       "; Single" : "; Married to " + princess.spouse->get_name() );



}

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

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << 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) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << 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) {}

void FrogPrince::display() const { cout << "Frog: " << name << endl;}

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

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

(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: " << 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) {}

void FrogPrince::display() const {
    cout << "Frog: " << name << endl;
}

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

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

(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: " << 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: " << name << endl;
}

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

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

(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: " << 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) {
    os << "Frog Prince: " << frog.name << endl;
}

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

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

(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: " << 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) {
    os << "Frog Prince: " << frog.name << endl;
    return os;
}

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

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

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
```