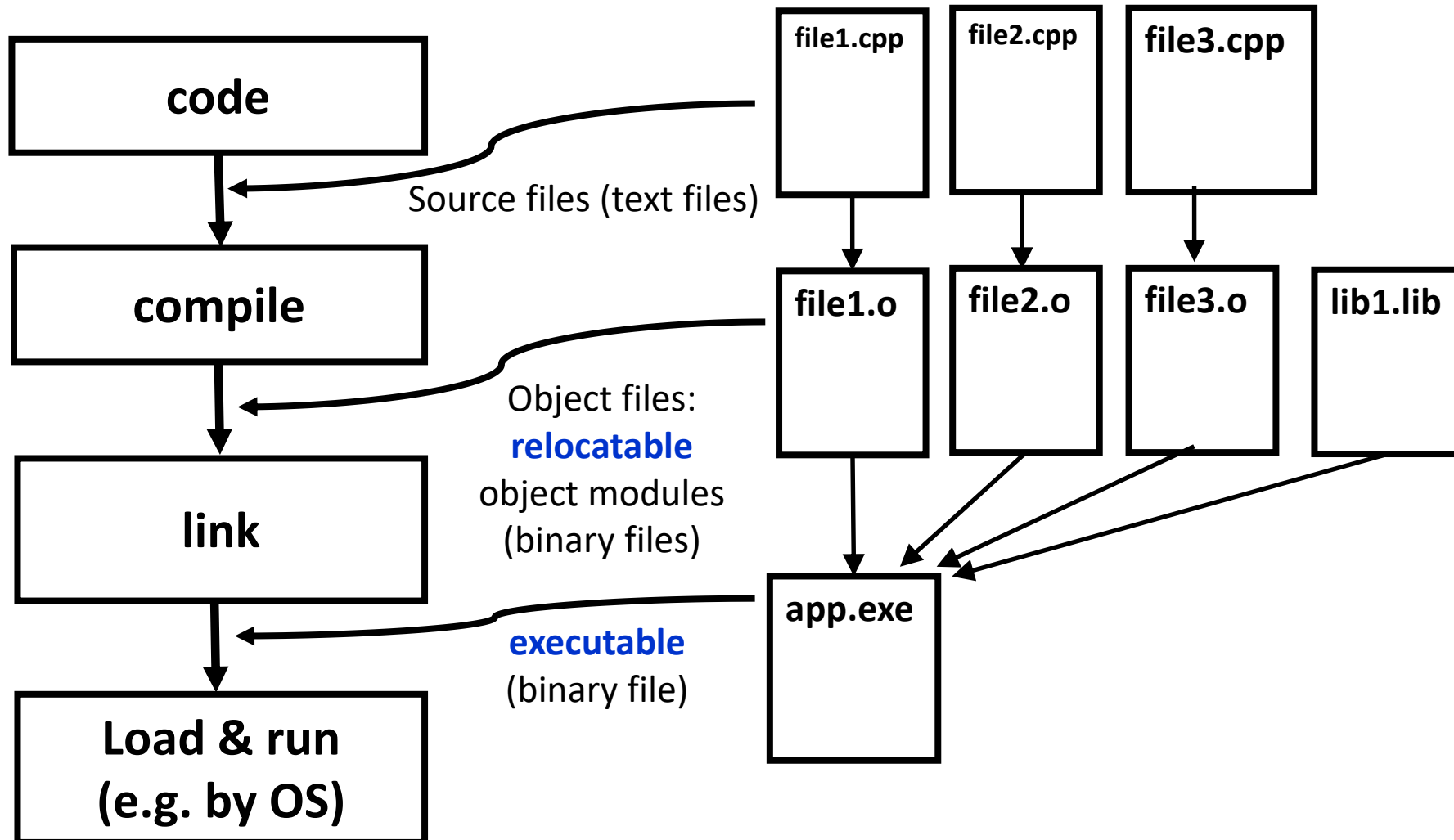
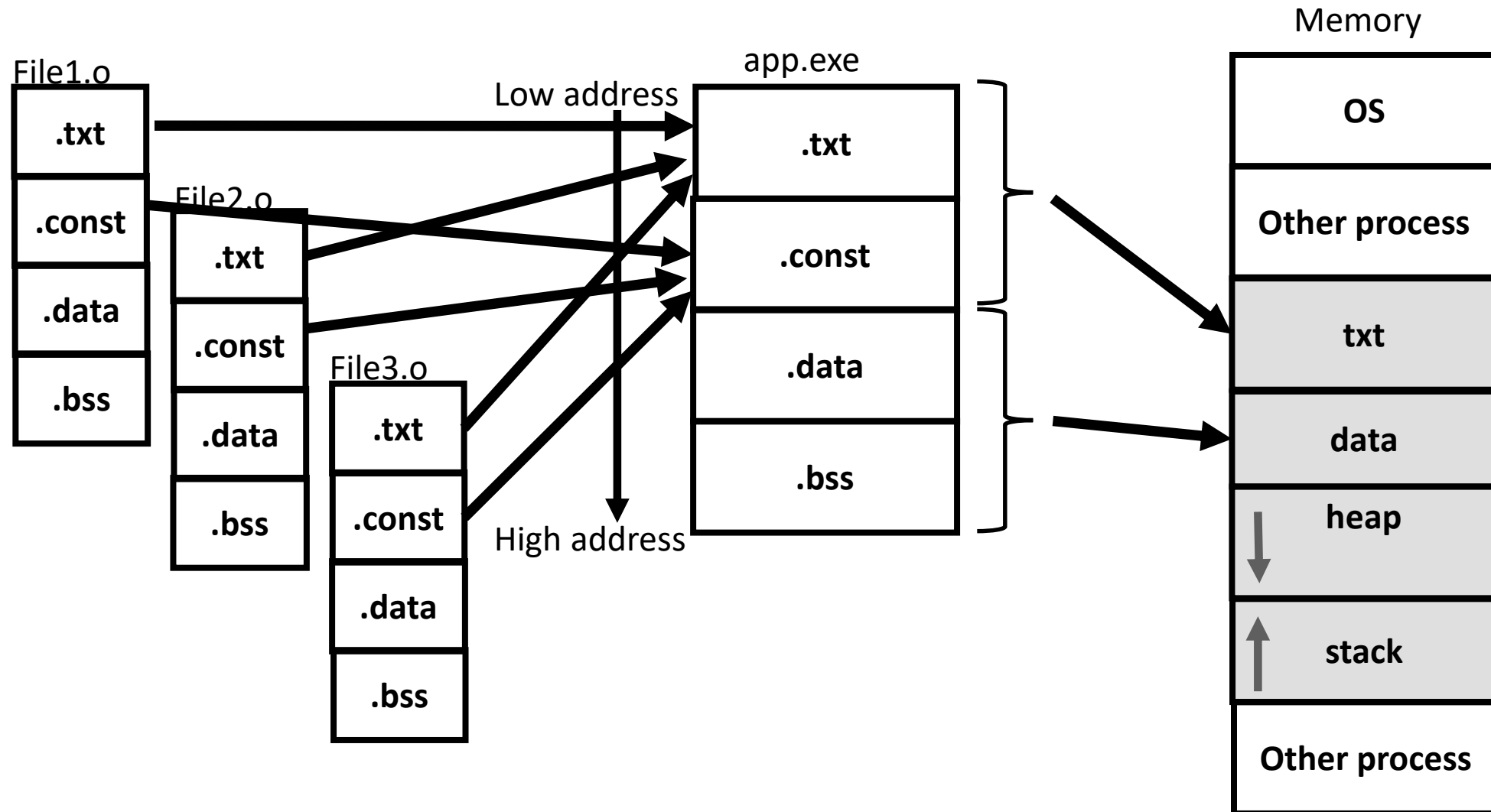


Program translation



Program translation – linking + loading



Program translation – linking + loading

Program translation involves:

- Compiling (source files → object files)
- Linking (putting together the executable: object files + libraries → executable)
- Loading and running (Loading your program: from hard drive → main memory):
 - Now your program is referred to as a “process”: A process is a program that is loaded and in execution.

Separate compilations – prince.cpp

```
// Prince.cpp
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {
    class Princess;

    class Prince {
        friend std::ostream& operator<<(std::ostream&, const Prince&);
    public:
        Prince(const std::string& name);
        void setSpouse(Princess* pp);
        const std::string& getName() const;
    private:
        std::string name;
        Princess* spouse;
    };

    // Prince implementation code
    Prince::Prince(const string& name) : name(name) {}

    void Prince::setSpouse(Princess* pp) {
        spouse = pp;
    }

    const string& Prince::getName() const { return name; }

    ostream& operator<<(ostream& os, const Prince& rhs) {
        os << "Prince: " << rhs.name;
        return os;
    }
}
```

Separate compilations – princess.cpp

```
// Princess.cpp
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {

    class Prince;

    class Princess {
    friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marries(Prince& aPrince);
    private:
        std::string name;
        Prince* spouse;
    };

    // Princess implementation code
    void Princess::marries(Prince& aPrince) {
        spouse = &aPrince;
        aPrince.setSpouse(this);
    }

    Princess::Princess(const string& name) : name(name) {}

    ostream& operator<<(ostream& os, const Princess& rhs) {
        os << "Princess: " << rhs.name;
        if (rhs.spouse != nullptr) {
            os << ", spouse: " << rhs.spouse->getName();
        }
        return os;
    }

}
```

Separate compilations – test.cpp

```
// test.cpp
#include <iostream>
#include <string>

using namespace std;
using namespace Fantasy;

int main() {
    Princess snowy("Snow White");
    cout << snowy << endl;
    Prince charmy("Charmy");
    cout << charmy << endl;
    snowy.marries(charmy);
    cout << snowy << endl;
}
```

- You may use:

```
g++ -std=c++14 -o test test.cpp Princess.cpp Prince.cpp
```

```
Build started...
1>----- Build started: Project: Lect_01_B, Configuration: Release x64 -----
1>Princess_om.cpp
1>Princess_om.cpp
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(23,18): error C2027: 'use
of undefined type 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(8): message : see
declaration of 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(32,45): error C2027: 'use
of undefined type 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(8): message : see
declaration of 'Fantasy::Prince'
1>test.cpp
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(6,24): error C2871: 'Fantasy': a
namespace with this name does not exist
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(9,5): error C2065: 'Princess':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(9,14): error C2146: syntax error:
missing ';' before identifier 'snowy'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(9,14): error C3861: 'snowy':
identifier not found
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(10,13): error C2065: 'snowy':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(11,5): error C2065: 'Prince':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(11,16): error C2146: syntax error:
missing ';' before identifier 'charmy'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(11,16): error C3861: 'charmy':
identifier not found
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(12,13): error C2065: 'charmy':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(13,5): error C2065: 'snowy':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(13,19): error C2065: 'charmy':
undeclared identifier
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(14,13): error C2065: 'snowy':
undeclared identifier
1>Done building project "cs2124.vcxproj" -- FAILED.
===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====
```

What is the problem?

- Source file `prince.cpp` does not have the declaration for class `Princess`.
- Source file `princess.cpp` does not have the declaration for class `Prince`.
- Source file `main.cpp` does not have the declaration for classes `Prince` and `Princess`.

Solution?

- Take the class declarations out of the source files (i.e. the .cp files) and place them in include files.
 - prince.h → declaration of Prince class
 - princess.h → declaration of Princess class
- Use

```
#include "filename"
```

in source files (.cpp files) to absorb the required class declarations

Header files – prince.h

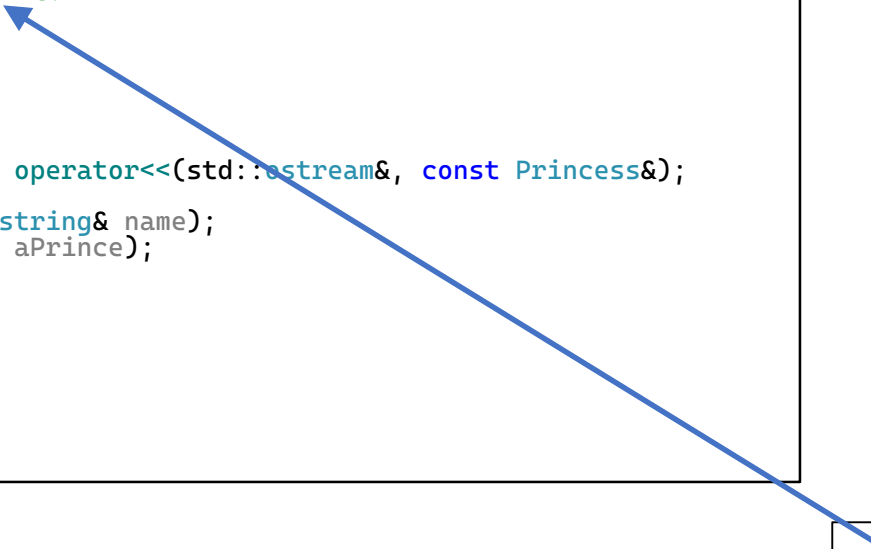
```
// Prince.h
#include <string>
#include <iostream>

namespace Fantasy {
    class Princess;
    class Prince {
        friend std::ostream& operator<<(std::ostream&, const Prince&);
    public:
        Prince(const std::string& name);
        void setSpouse(Princess* pp);
        const std::string& getName() const;
    private:
        std::string name;
        Princess* spouse;
    };
}
```

Header files – princess.h

```
//Princess.h
#include <string>
#include <iostream>

//using namespace std; // Don't!
namespace Fantasy {
    class Prince;
    class Princess {
        friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marries(Prince& aPrince);
    private:
        std::string name;
        Prince* spouse;
    };
}
```



- Do NOT use “using namespace ..” in header files.

Source files – prince.cpp

```
// Prince.cpp
#include "Prince.h"
#include "Princess.h"
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {

    // Prince implementation code
    Prince::Prince(const string& name) : name(name) {}

    void Prince::setSpouse(Princess* pp) {
        spouse = pp;
    }

    const string& Prince::getName() const { return name; }

    ostream& operator<<(ostream& os, const Prince& rhs) {
        os << "Prince: " << rhs.name;
        return os;
    }

}
```

Source files – princess.cpp

```
// Princess.cpp
#include "Princess.h"
#include "Prince.h"
#include <string>
#include <iostream>
using namespace std;

namespace Fantasy {

    void Princess::marries(Prince& aPrince) {
        spouse = &aPrince;
        aPrince.setSpouse(this);
    }

    Princess::Princess(const string& name) : name(name) {}

    ostream& operator<<(ostream& os, const Princess& rhs) {
        os << "Princess: " << rhs.name;
        if (rhs.spouse != nullptr) {
            os << ", spouse: " << rhs.spouse->getName();
        }
        return os;
    }

}
```

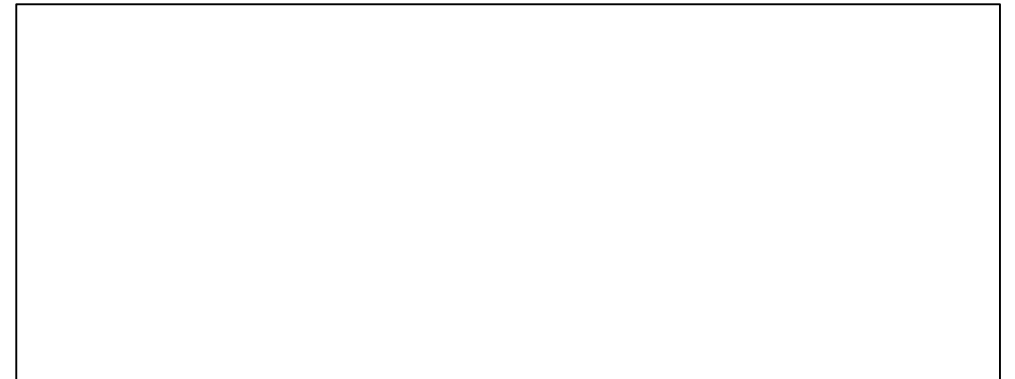
Source files – test.cpp

```
// test.cpp
#include <iostream>
#include <string>
#include "Prince.h"
#include "Princess.h"

using namespace std;
using namespace Fantasy;

int main() {
    Princess snowy("Snow White");
    cout << snowy << endl;
    Prince charmy("Charmy");
    cout << charmy << endl;
    snowy.marries(charmy);
    cout << snowy << endl;
}
```

Princess: Snow White
Prince: Charmy
Princess: Snow White, spouse: Charmy



Notes on header files

- Avoid the “using” keyword in header files – why?
 - A source file may include multiple header files, each with a different namespace → this causes namespace pollution (and collisions of course)
 - Should leave that decision (of which namespace to use) to a source file (i.e. a .cpp file)
 - Please recall that we cannot “`un-use namespace xyz`”, there’s no such syntax.
- Remember to scope the standard library types (i.e. `std::type`), even though you normally wouldn’t scope in a source file (since “using” is commonly used inside source files).
- Avoid cyclic dependencies: Our two header files shall not both include each others. One header file (`princess.h`) does a forward declaration, whereas the second header file (`prince.h`) may include the first.

Header files – multiple or cyclic includes – cont.

```
//Princess.h
#include <string>
#include <iostream>
#include "Prince.h"
//using namespace std; // Don't!

namespace Fantasy {
    //class Prince;

    class Princess {
    friend std::ostream& operator<<(std::ostream&, const Princess&);
    public:
        Princess(const std::string& name);
        void marries(Prince& aPrince);
    private:
        std::string name;
        Prince* spouse;
    };
}
```

When compiling princess.cpp:
prince.h is now included
MULTIPLE TIMES

```
Build started...
1>----- Build started: Project: Lect_01_B, Configuration: Release x64 -----
1>Prince_om.cpp
1>Princess_om.cpp
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Prince.h(9,22):
error C2011: 'Fantasy::Prince': 'class' type redefinition
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess.h(9):
message : see declaration of 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(12,18): error C2027: use of undefined type 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess.h(9):
message : see declaration of 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess_om.cpp(21,45): error C2027: use of undefined type 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess.h(9):
message : see declaration of 'Fantasy::Prince'
1>test.cpp
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Prince.h(9,22):
error C2011: 'Fantasy::Prince': 'class' type redefinition
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Prince.h(9):
message : see declaration of 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(13,16):
error C2079: 'charmy' uses undefined class 'Fantasy::Prince'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(13,31):
error C2440: 'initializing': cannot convert from 'const char [7]' to 'int'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(13,22):
message : There is no context in which this conversion is possible
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\test.cpp(15,25):
error C2664: 'void Fantasy::Princess::marries(Fantasy::Prince &)': cannot convert argument 1
from 'int' to 'Fantasy::Prince &'
1>C:\Dropbox\CS2124_OOP_2023_Spring\lect_code\05b.Separate Compilation\Princess.h(15,14): message : see declaration of 'Fantasy::Princess::marries'
1>Done building project "cs2124.vcxproj" -- FAILED.
===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====
```

- **Include guards** may be used to avoid cycling includes and redefinitions

Notes on header files – cont.

- Add include guards to make sure a header file “prince.h” is not included multiple times

```
#ifndef PRINCE_H
#define PRINCE_H

.
<header file's body>
.
.
#endif
```


Include guards

```
// Prince.h
#include <string>

#ifndef PRINCE_H
#define PRINCE_H
namespace Fantasy {
    class Princess;
    class Prince {
        friend std::ostream& operator<<(std::ostream&, const Prince&);
    public:
        Prince(const std::string& name);
        void setSpouse(Princess* pp);
        const std::string& getName() const;
    private:
        std::string name;
        Princess* spouse;
    };
}
#endif
```

Princess: Snow White
Prince: Charmy
Princess: Snow White, spouse: Charmy