Headers and Testing

EECS 211

Winter 2017

Declarations

A declaration introduces a *name* into a *scope* (region of code):

- gives a type for the named object
- sometimes includes an initializer
- must come before use

Declarations

A declaration introduces a *name* into a *scope* (region of code):

- gives a type for the named object
- sometimes includes an initializer
- must come before use

Examples:

- `int a = 7;`
- `int b;`
- `vector<string> c;`
- `double my_sqrt(double);`

Headers

Declarations are frequently introduced through *headers*:

```
int main()
{
std::cout << "Hello, world!\n";
}
```

Error: unknown identifier std::cout

Headers

Declarations are frequently introduced through *headers*:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
int b;        // but why?
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
int b;          // but why?
vector<double> v;
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
int b;        // but why?
vector<double> v;
double square(double x) { return x * x; }
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
int b;          // but why?
vector<double> v;
double square(double x) { return x * x; }
struct Point { int x, y; };
```

Definitions

A declaration that (also) fully specifies the declarandum is a *definition*.

Examples:

```
int a = 5;
int b;          // but why?
vector<double> v;
double square(double x) { return x * x; }
struct Point { int x, y; };
```

Examples of non-definition declarations:

```
extern int b;
double square(double);
struct Point;
```

Declarations and definitions

	declarations	definitions
may be repeated	yes	no
must come before use	yes	no

Why both?

To refer to something, we need only its declaration

We can hide its definition, or save it for later

In large programs, declarations go in header files to ease sharing

Declaration example

```
double my_sqrt(double x)
{
        ⋮
}

int main()
{
        ··· my_sqrt(y) ···
}
```

Declaration example

```
int main()
{
        ··· my_sqrt(y) ···          // unknown identifier
}

double my_sqrt(double x)
{
        ⋮
}
```

Declaration example

```
double my_sqrt(double);

int main()
{
··· my_sqrt(y) ···
}

double my_sqrt(double x)
{
      ⋮
}
```

Library declaration example

In `my_math.h`:

```
double my_sqrt(double);
```

In `my_math.cpp`:

```
#include ”my_math.h”

double my_sqrt(double x)
{ ··· }
```

In some other (*client*) `.cpp` source file:

```
#include ”my_math.h”

int f() { ··· my_sqrt(c) ··· }
```

Testing

One client of our library code is our test suite, in `my_math_test.cpp`:

```
#include ”my_math.h”
#include <UnitTest++/UnitTest++.h>

TEST(My_sqrt_9_is_correct)
{
    CHECK_EQUAL(3, my_sqrt(9));
}
```

More testing

```
#include ”my_math.h”
#include <UnitTest++/UnitTest++.h>

TEST(My_sqrt_2_is_close)
{
    CHECK_CLOSE(1.414, my_sqrt(2), 0.001);
}

TEST(My_sqrt_throws_on_negative)
{
    CHECK_THROW(my_sqrt(-9), std::runtime_error);
}
```

Building

`CMakeLists.txt` needs to specify which files should be compiled together to make which programs:

```
cmake_minimum_required(VERSION 3.3)
project(my_sqrt CXX)
include(.eecs211/CMakeLists.txt)

add_program(sqrt_client
    sqrt_client.cpp
    my_sqrt.cpp)

add_test_program(my_sqrt_test
    my_sqrt_test.cpp
    my_sqrt.cpp)
```

– To CLion! –