Language Basics – User Defined Types

Kate Gregory

@gregcons www.gregcons.com/kateblog

Building

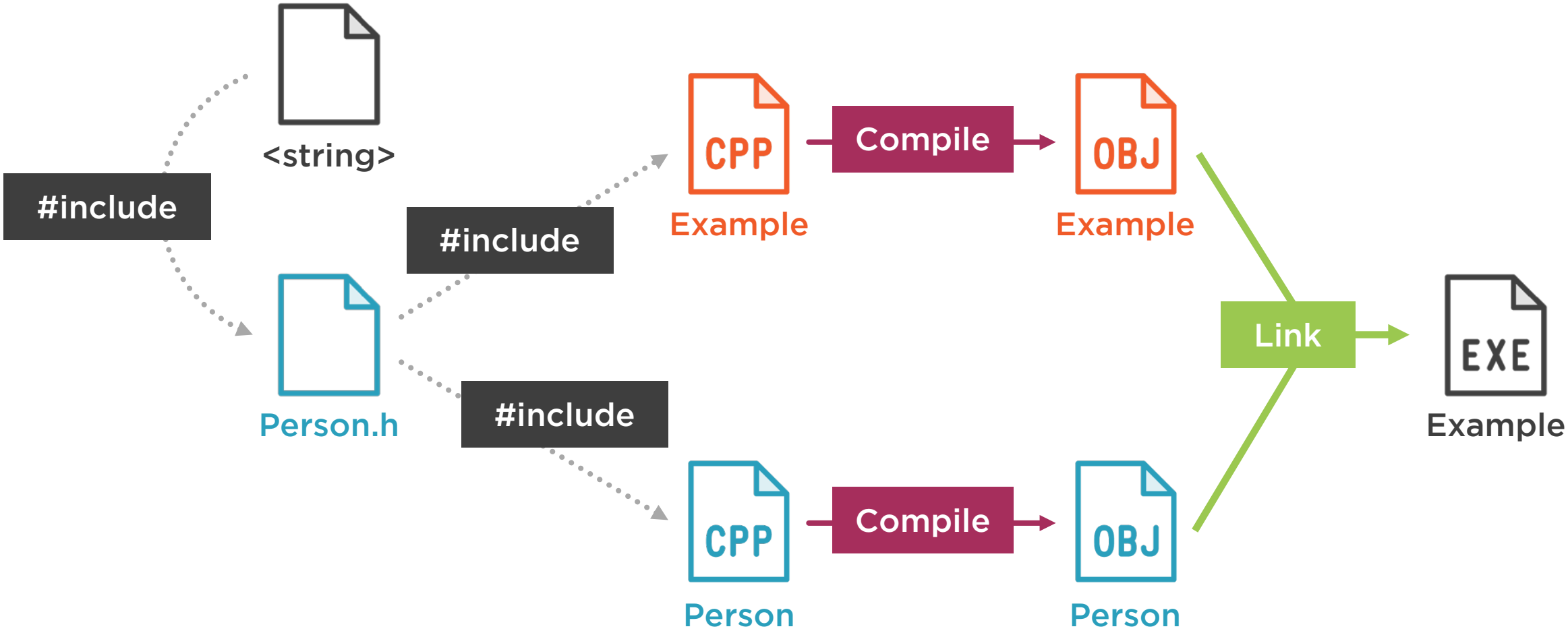
You can make the compiler a promise

Linker will ensure it is kept

#include only a convenience

Put each class in its own file

Building


<string>
#include
#include
#include
Person.h
CPP
Example
Compile
OBJ
Example
CPP
Person
Compile
OBJ
Person
Link
EXE
Example

Classes and Objects

`class` keyword

private and public sections

Declare like fundamental types

Access members with .

```
class Person
{
private:
    std::string firstname;
    std::string lastname;
    int arbitrarynumber;

public:
    std::string getName();
};
```

- The keyword class introduces the declaration
- Brace brackets surround the contents
- Member variables are generally private

- Functions are generally public
- Don’t have to code them, just declare them

- Don’t forget the semi-colon

```
#include "Person.h"


std::string Person::getName()
{
    return firstname + " " +
lastname;
}
```

- Includes the same header as consuming code uses

- Use fully qualified name
- Body surrounded by braces

- Access member variables with no special syntax

Objects

Instance of a class

Has own copy of member variables

Member functions operate on member variables (and arguments)

Initialized with a constructor

```
class Person
{
// . . .
    Person(std::string first,
        std::string last,
        int arbitrary);
// . . .
};
```

- Constructors have the same name as the class
- They take arguments, which have a type and a name
- They do not return a value

```
Person::Person(std::string first,
               std::string last,
               int arbitrary)

       :

firstname(first),
lastname(last),
arbitrarynumber(arbitrary)

{

}
```

- Use fully qualified name
- No return type
- This colon introduces initializers
- Braces around the body
- Empty body is quite normal

Scope

Have a lifetime

Constructor

Scope

Destructor

Resource Acquisition Is Initialization

Acquire in constructor

Release in destructor

struct

"plain old data"

member functions, constructor, destructor

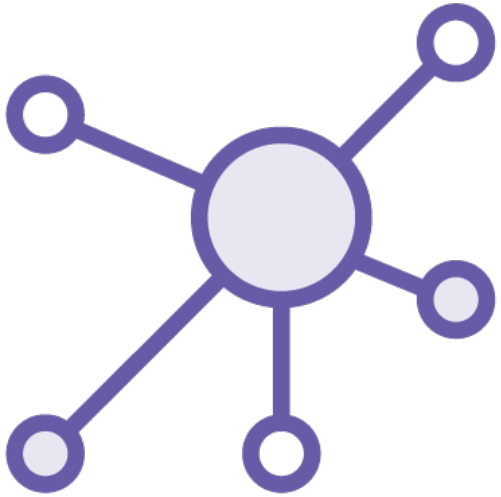
default access is public

Namespaces

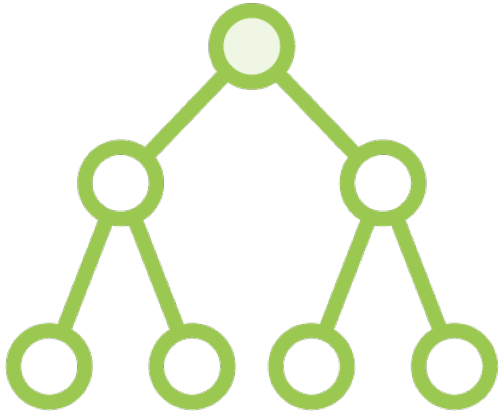
Prevent name collisions

Separate from class name with ::

Inheritance



Key to OO Design



Derived classes

Add or override

```
class Tweeter :
    public Person
{

};
```

- Same class keyword
- Colon indicates inheritance
- List only additional member variables and functions

```
Tweeter::Tweeter(
        std::string first,
        std::string last,
        int arbitrary,
        std::string handle)

    :

    Person(first,last,
           arbitrary),

    twitterhandle(handle)

{

}
```

- Pass some parameters to the base class in the initialization

Enum

Give names to a set of constants

Names must be unique

```
enum Status
{
    Pending,
    Approved,
    Cancelled
};
```

- Keyword enum
- Braces surround the entries
- Comma after each one

- No comma after the last
- Don’t forget the semi-colon

Scoped Enum

Allows underlying type other than int

Names don't have to be unique

Use fully qualified names

```
enum class FileError
{
    notfound,
    ok
};
enum class NetworkError
{
    disconnected,
    ok
};
```

- Keyword enum class
- Braces surround the entries
- Comma after each one
- Don’t forget the semi-colon
- Two “ok”s is ok

```
Status s = Pending;
s = Approved;


FileError fe =
   FileError::notfound;

fe = FileError::ok;

NetworkError ne =
   NetworkError::disconnected;

ne = NetworkError::ok;
```

- Declare as though the enum was a type

- For scoped enums, fully qualify the enum entry

PreProcessor

Lines start #

Control what is compiled

Use with care

Summary

Declare instances of objects or fundamental types on the stack

When the instance goes out of scope, the object is cleaned up

User defined types and fundamental types are equally real

Scoped enums make it simple to use named values instead of arbitrary constants