This is 183

L14: Week 9 - Monday

Reminders

- Assignment 4 due Friday!

Final Projects + Showcase

Choice of 4 projects:

- Connect 4
- Creative AI
- Micro-Arcade
- Web Scheduler


Choice of 4 projects:
– Connect 4
– Creative AI
– Micro-Arcade
– Web Scheduler

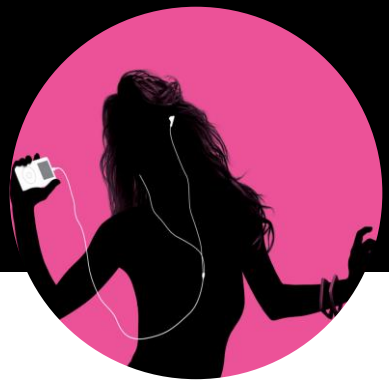
Creative AI (python)

Machine Learning

Use machine learning, which is a subset of artificial intelligence, to write programs that learn from and act on data.

NLP

Learn about Natural Language Processing, which involves computers interacting with human languages.



Lyrics + Music

Generate new lyrics and playable music using datasets of your choice.

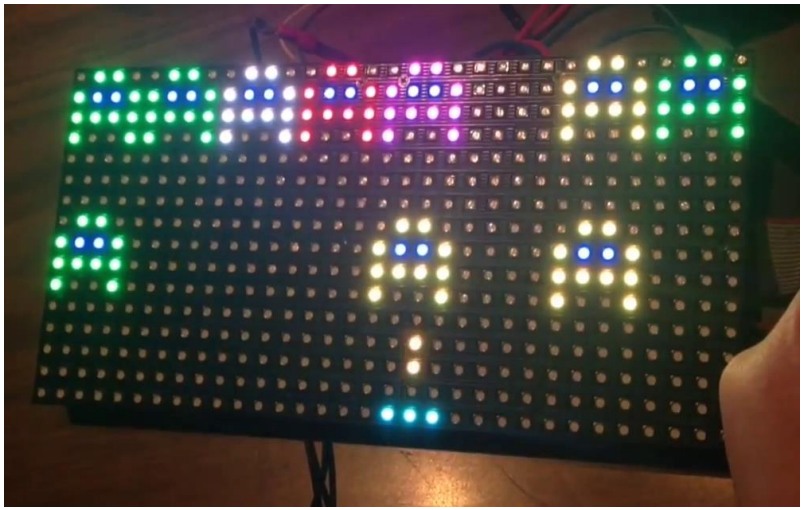
Endless Possibilities

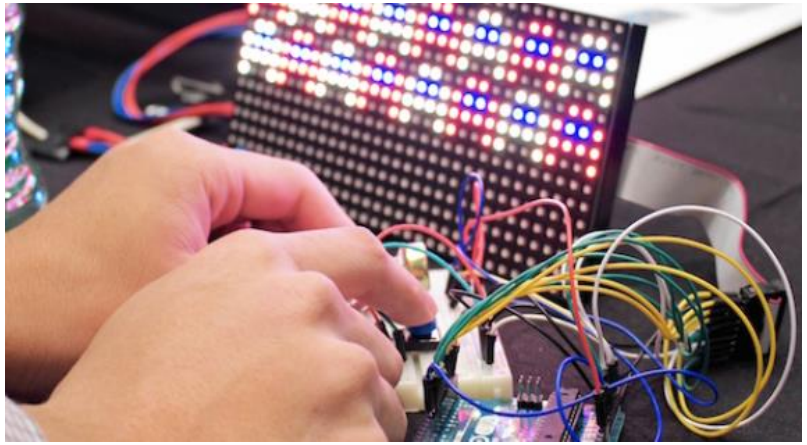
Write a program that will generate unique lyrics and music to fit your creative visions.

[Lyric Generation Demo]

[Music Generation Demo]

Arduino: Micro-Arcade (C++)



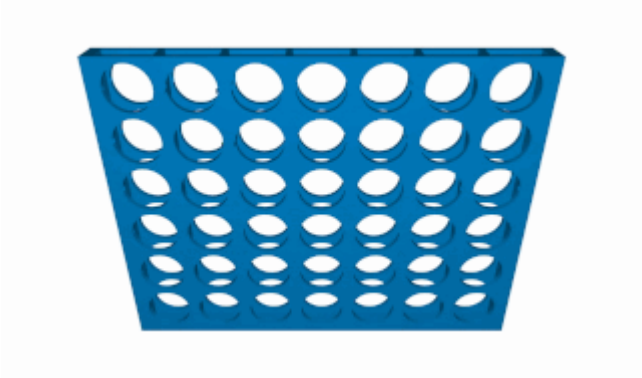


- Gain experience with **hardware** and **engineering**
- Watch your **game implementation** come to life on a beautiful LED board
- Learn about game design
- Have **fun** playing games

Demo

Connect 4

Classic game of Connect 4



C++ && OpenGL for graphics

Write AI for computer

Demo

Web Scheduler

- Create a **website** to search and view classes
- Opportunity for **creativity** in design and features
- Develop in Python and HTML
- Deploy using **cloud computing** infrastructure
- Gain **valuable skills** for future employers
- Create the next University scheduler!

Planitt GoBlueprint MyCourseGuide PlanBlue VeryGoodScheduler


SNICKERS


EXIT
CODE SOMETHING
Epic


BALLROOM
EECS 183
Showcase
Platinum Sponsor
JPMorgan Chase & Co.
Gold Sponsor
stryker
Silver Sponsors
Epic
wolverine
Start-Up Sponsors
FarmLogs
akuna capital
WorkForce Software
Microsoft

Final Project Logistics

- Groups of 4 – **no exceptions**
  - Can be from different lectures/discussions
  - Use the teammates thread on Piazza
  - Next class, you will be given time to find teammates interested in the same project

- We need to know your team and your project by **3/25**

Final Project Logistics

- If (auditing the course, or taking pass/fail) and (don't need to or
  don't want to do the final project),
  please don't sign up for a team.

Final Project Logistics

- If you want to drop the course, **you MUST do it by 3/18.**

- No exceptions –
- We will **not** sign drop form, change of status form after 3/18, regardless of your school policy.

Saline Girls Who Code Club welcomes

Chuck Hess

senior director of engineering &
CTO of the slots division for Zynga

Monday, March 7th at 6:30 Saline HS Media Center

Anyone interested in coding is invited. Chuck will talk about his programming background and life as a CTO at Zynga.

girls who CODE CLUBS

zynga
CONNECTING THE WORLD THROUGH GAMES

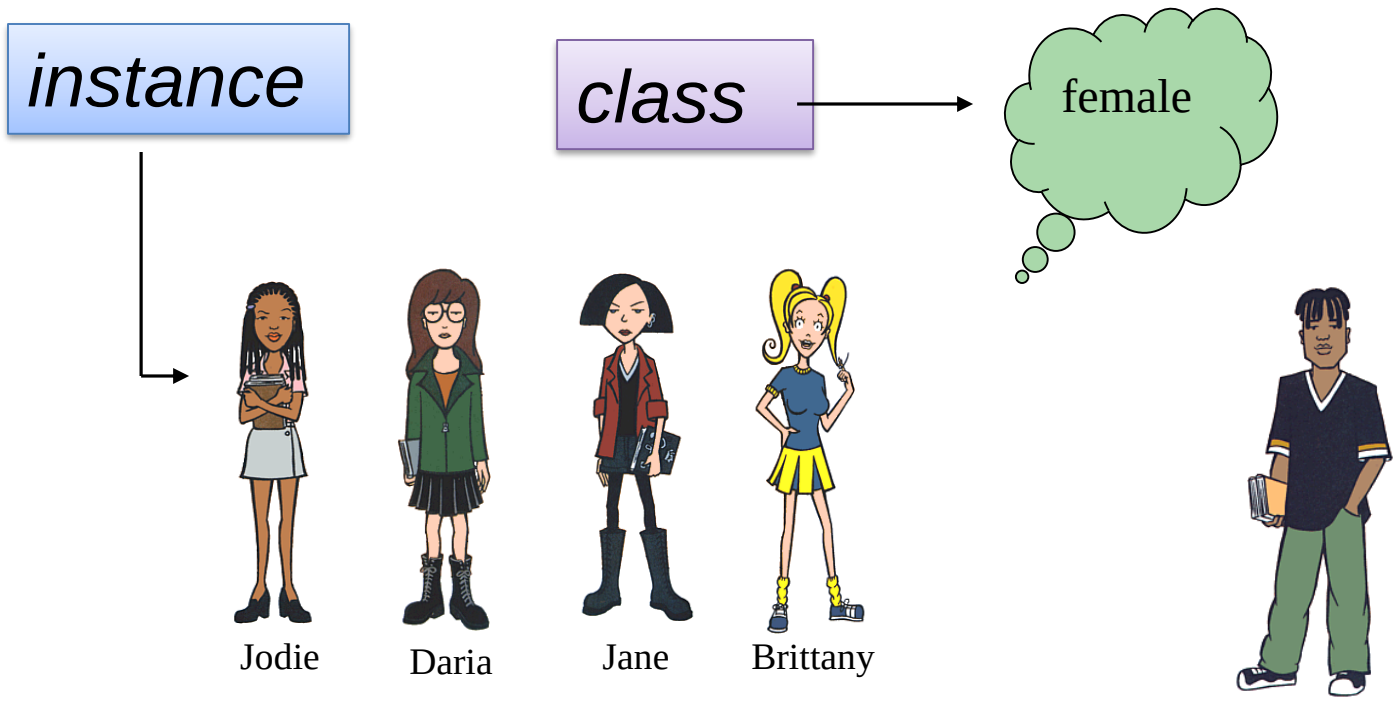
Last Time… on EECS 183

Concept of class's

Custom Data Types

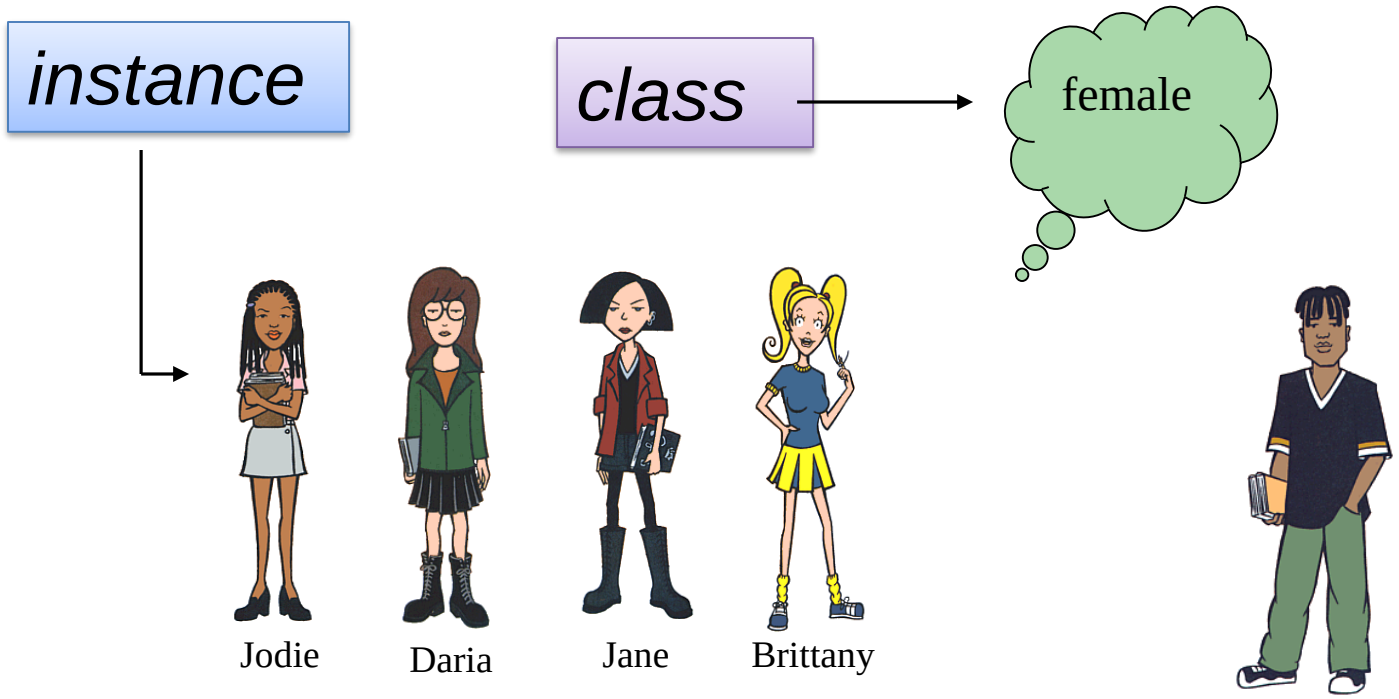
Review

class vs. instance of class



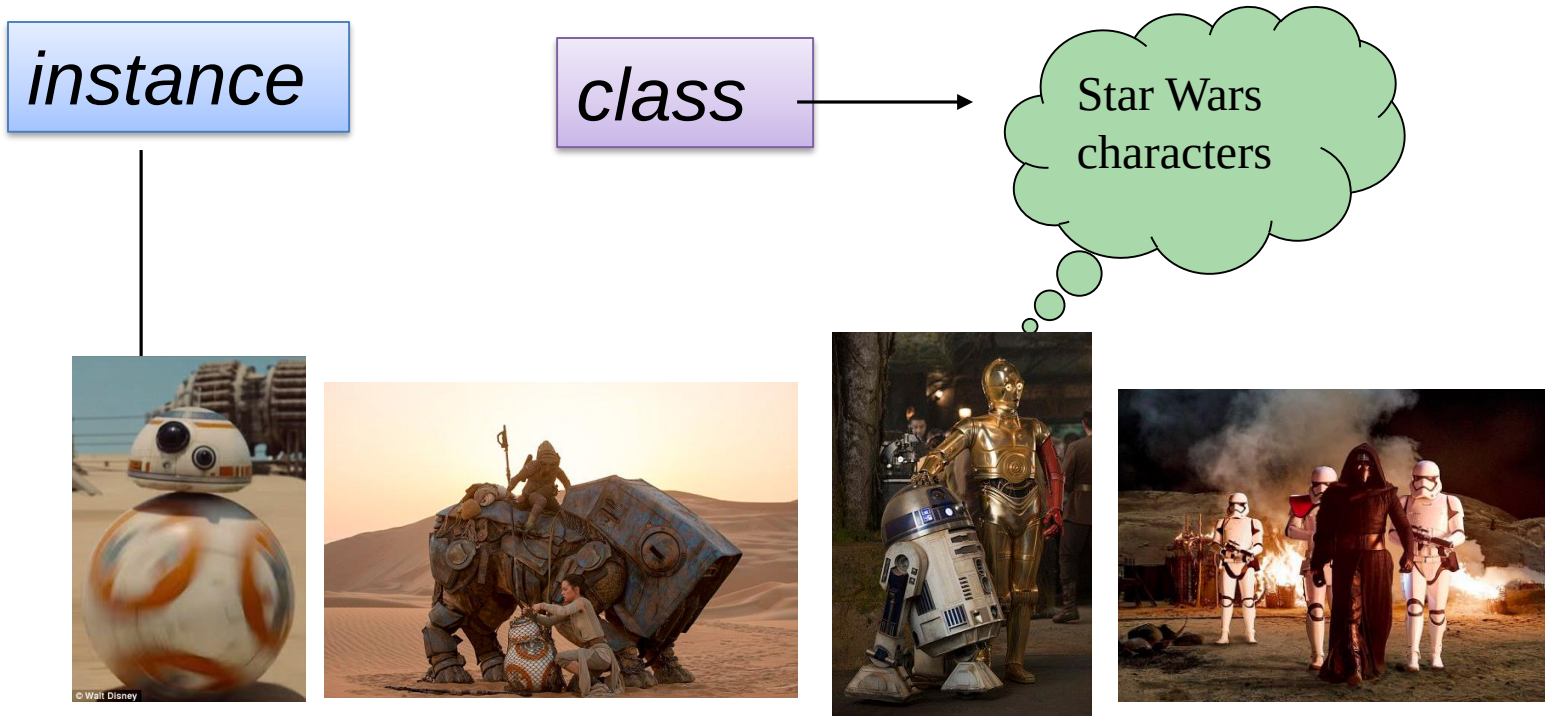
class vs. instance of class

- Classes reflect concepts
- Instance is a single and unique unit of the class



class vs. instance of class

- Classes reflect concepts
- Instance is a single and unique unit of the class



i>clicker #1

Given:

```
ifstream file;
file.open("final.txt");
```

Which of the following are true?

A) `ifstream` is a class, `file` is a class
B) `ifstream` is an instance, `file` is a class
C) `ifstream` is a class, `file` is an instance
D) `ifstream` is an instance, `file` is an instance
E) None of the above

i>clicker #1

Given:

```
ifstream file;
file.open("final.txt");
```

Which of the following are true?

A) `ifstream` is a class, `file` is a class
B) `ifstream` is an instance, `file` is a class
C) `ifstream` is a class, `file` is an instance
D) `ifstream` is an instance, `file` is an instance
E) None of the above

i>clicker #2

Given:

```
string str;
```

Which of the following is NOT true?

A) str is an identifier
B) str is an instance
C) str is a variable
D) str is a class
E) all the above are true

i>clicker #2

Given:

```
string str;
```

Which of the following is NOT true?

A) str is an identifier
B) str is an instance
C) str is a variable
D) str is a class
E) all the above are true

i>clicker #3

Given:

```
string str;
str = "A long time ago in a galaxy far, far away";
int length = str.length();
```

length() in str.length() is

A) a function
B) a member function
C) a mistake

i>clicker #3

Given:

```
string str;
str = "A long time ago in a galaxy far, far away";
int length = str.length();
```

length() in str.length() is

A) a function

B) a member function

C) a mistake

Summary of Last Time

- **Classes** group together
  - Heterogeneous data
  - Relevant functions

- We can have
  - one class (`Student`) and
  - multiple **instances** (`kevin_lee`, `maddy_endres`, etc.)

- Benefits of classes:
  1. Keeps related data and functionality together
  2. Separate interface from implementation
  3. Avoid code duplication

Today

code

The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The De

This is a C++ keyword, to signify a class definition.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The D

This class is called Student; by convention, class names start in Upper Case

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

T

This is another C++ keyword. means accessible from anywhere where the object is visible.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

All students have a `first_name`, which is a `string`. This is called a **member variable**

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The D

All students also have a last_name, which is also a string.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The

Note the semicolon. This is actually a statement telling C++ that a Student class exists

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The D

Create an *instance* of Student called kevin.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Set the first_name and last_name of kevin to "Kevin" and "Lee" respectively.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Get the first_name of kevin.

The Details

since "first_name" is public, can access here.

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Get the last_name of kevin.

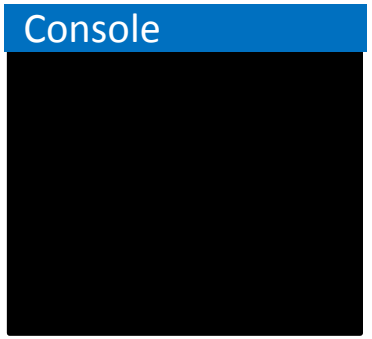
The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Console



The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Console

```
Kevin Lee
```

The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin;
    kevin.first_name = "Kevin";
    kevin.last_name = "Lee";
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Console

```
Kevin Lee
```

i>Clicker #4

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student helen = {"Maddy", "Endres"};
    helen.first_name = "Kevin";
    helen.last_name = "Lee";
    cout << helen.first_name << " "
         << helen.last_name << endl;
}
```

What prints?

A. Helen Hagos
B. Maddy Endres
C. Kevin Lee
D. Something else
E. Compile error

i>Clicker #4

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student helen = {"Maddy", "Endres"};
    helen.first_name = "Kevin";
    helen.last_name = "Lee";
    cout << helen.first_name << " "
         << helen.last_name << endl;
}
```

What prints?

A. Helen Hagos
B. Maddy Endres
C. Kevin Lee
D. Something else
E. Compile error

The Details

```
class Student {
    public:
        string first_name;
        string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Console

```
Kevin Lee
```

Member Functions

This **member function** is just like the functions you already know

```
class Student {
    public:
        string first_name;
        string last_name;
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    kevin.print_name();
}
```

Member Functions

```
class Student {
    public:
        string first_name;
        string last_name;
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    kevin.print_name();
}
```

But it's called with a '.', like how we call `inFile.open("filename");`

Member Functions

Notice that first_name can be used directly, since it's in the *scope* of the Student class

```
class Student {
    public:
        string first_name;
        string last_name;
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    kevin.print_name();
}
```

Member Functions

```
class Student {
    public:
        string first_name;
        string last_name;
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    kevin.print_name();
}
```

Console

```
Kevin Lee
```

Public and Private Members

```
class Student {
    public:
        string first_name;
        string last_name;
        double gpa;
};

int main() {
    Student kevin = {"Kevin", "Lee", 1.3};

    cout << kevin.gpa << endl;
}
```

Console

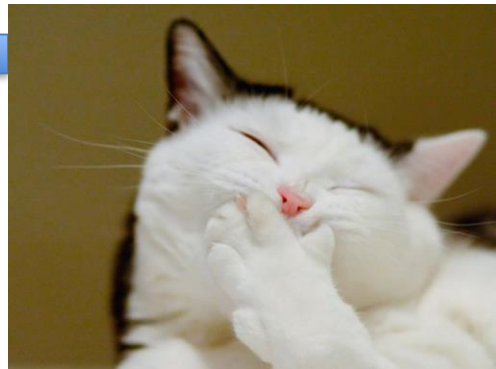
```
1.3
```

Public and Private Members

```
class Student {
    public:
        string first_name;
        string last_name;
        double gpa;
};

int main() {
    Student kevin = {"Kevin", "Lee", 1.3};
    kevin.gpa = 4.0;
    cout << kevin.gpa << endl;
}
```

Console

```
4.0
```



Public and Private Members

Private is another C++ keyword.
accessible only from member functions

```
class Student {
    private:
        string first_name;
        string last_name;
        double gpa;
};

int main() {
    Student kevin = {"Kevin", "Lee", 1.3};
    kevin.gpa = 4.0;
    cout << kevin.gpa << endl;
}
```

Public and Private Members

```
class Student {
    private:
        string first_name;
        string last_name;
        double gpa;
};

int main() {
    Student kevin = {"Kevin"
    kevin.gpa = 4.0;
    cout << k
}
```

Compile Error

nd

Constructors

```
class Student {
    private:
        string first_name;
        string last_name;
        double gpa;
};

int main() {
    Student kevin = {"Kevin", "Lee", 1.3};

    cout << kevin.gpa << endl;
}
```

Constructors

Constructors

- A **Constructor** is a member function that **has the same name as the class** and **no return type**.

- It's what creates the instance of the class.

Constructors

```
class Student {
    public:
        Student(string first, string last) {
            first_name = first;
            last_name = last;
        }
    private:
        string first_name;
        string last_name;
};
int main() {
    Student kevin("Kevin", "Lee");
}
```

This constructor sets the first and last name of a student.

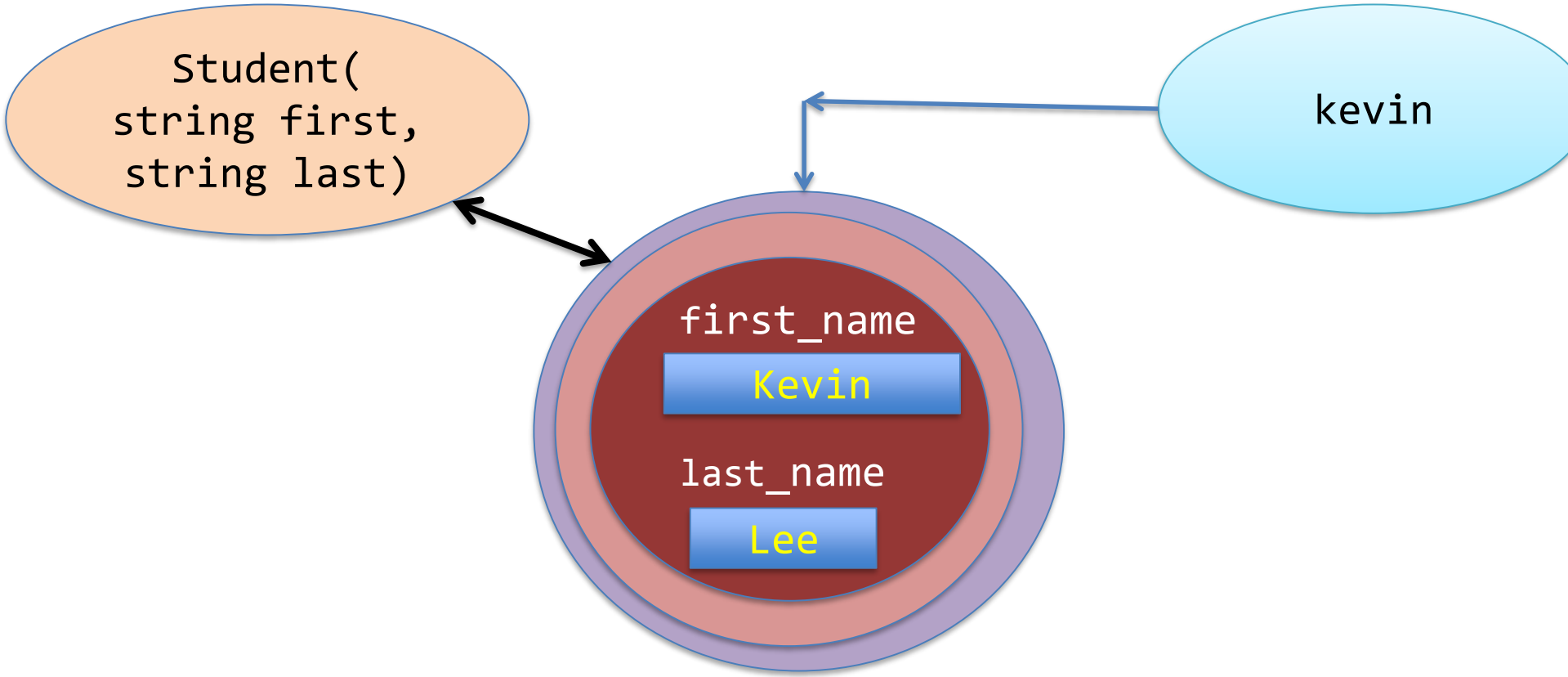
Constructors

We "call" the function when we initialize kevin.

```
class Student {
    public:
        Student(string first, string last) {
            first_name = first;
            last_name = last;
        }
    private:
        string first_name;
        string last_name;
};
int main() {
    Student kevin("Kevin", "Lee");
}
```

Visual

Separates interface from implementation




```
Student kevin("Kevin", "Lee");
```

Getters

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
    private:
        string first_name;
        string last_name;
};
int main() {
    Student kevin("Kevin", "Lee");
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Getters

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
    private:
        string first_name;
        string last_name;
};
int main() {
    Student kevin("Kevin", "Lee");
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Compile Error

Member Functions

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
    private:
        string first_name;
        string last_name;
};
```

Getters

Private
accessible only from member functions

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
    private:
        string first_name;
        string last_name;
};
```

Get

Member functions can always access private

Private
accessible only from member functions

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
        void print_name() {
            cout << first_name << " "
                 << last_name << endl;
        }
    private:
        string first_name;
        string last_name;
};
```

Getters

A **getter** is a function that returns the value in a private variable.

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
        string get_first_name() {
            return first_name;
        }
    private:
        string first_name;
        string last_name;
};
```

Getters

A **getter** is a function that returns the value in a private variable.

```
class Student {
    public:
        Student(string first,
                string last) {
            first_name = first;
            last_name = last;
        }
        string get_first_name() {
            return first_name;
        }
        string get_last_name() {
            return last_name;
        }
    private:
        string first_name;
        string last_name;
};
```

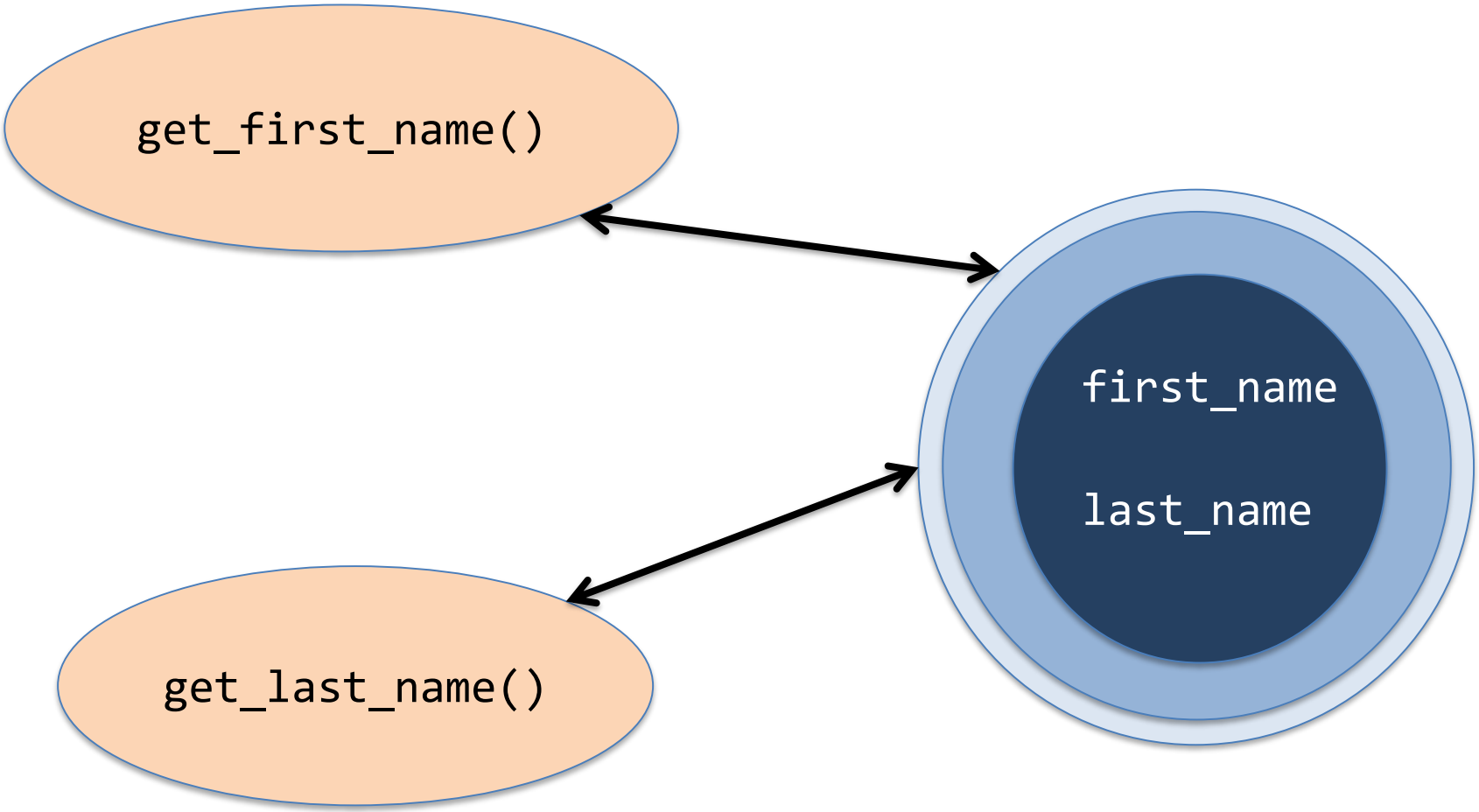
Getters

```
class Student {
    public:
        ...
        string get_first_name() {
            return first_name;
        }
        string get_last_name() {
            return last_name;
        }
    ...
};

int main() {
    Student kevin("Kevin", "Lee");
    cout << kevin.get_first_name() << " "
         << kevin.get_last_name() << endl;
}
```

Console

```
Kevin Lee
```

Visualization



i>Clicker #5

```
class Student {
    public:
        Student(string name,
                int g) {
            full_name = name;
            gpa = g;
        }
    private:
        string full_name;
        double gpa;
};

int main() {
    Student maddy("Madeline Endres", 4.0);
    maddy.gpa = 3.5;
    cout << maddy.gpa << endl;
}
```

What does this program do?

A. Prints 4.0
B. Prints 3.5
C. There will be a compile error
D. There will be a runtime error
E. None of the above

i>Clicker #5

```
class Student {
    public:
        Student(string name,
                int g) {
            full_name = name;
            gpa = g;
        }
    private:
        string full_name;
        double gpa;
};

int main() {
    Student maddy("Madeline Endres", 4.0);
    maddy.gpa = 3.5;
    cout << maddy.gpa << endl;
}
```

What does this program do?

A. 4.0
B. Prints 3.5
C. There will be a compile error
D. There will be a runtime error
E. None of the above

i>Clicker #6

```
class Student {
    public:
        Student(string name,
                int g) {
            full_name = name;
            gpa = g;
        }
    private:
        string full_name;
        double gpa;
        void print_gpa() {
            cout << full_name << ": " << gpa << endl;
        }
};

int main() {
    Student maddy("Madeline Endres", 4.0);
    maddy.print_gpa();
}
```

What does this program do?

A. Prints Madeline Endres: 4.0
B. There will be a compile error
C. There will be a runtime error
D. None of the above

i>Clicker #6

```
class Student {
    public:
        Student(string name,
                int g) {
            full_name = name;
            gpa = g;
        }
    private:
        string full_name;
        double gpa;
        void print_gpa() {
            cout << full_name << ": " << gpa << endl;
        }
};

int main() {
    Student maddy("Madeline Endres", 4.0);
    maddy.print_gpa();
}
```

What does this program do?

A. Prints Madeline Endres: 4.0
B. There will be a compile error
C. There will be a runtime error
D. None of the above

i>Clicker #6

```
class Student {
    public:
        Student(string name,
                int g) {
            full_name = name;
            gpa = g;
        }
        void print_gpa() {
            cout << full_name << ": " <<
        }

    private:
        string full_name;
        double gpa;
};

int main() {
    Student maddy("Madeline Endres", 4.0);
    maddy.print_gpa();
}
```

What does this program do?

A. Prints Madeline Endres: 4.0
B. There will be a compile error
C. There will be a runtime error
D. None of the above

Setters

Sometimes you'll also see **setters**, which change private variables.

```
class Student {
    public:
        ...
        string get_full_name() {
            return full_name;
        }
        void set_full_name(string name) {
            full_name = name;
        }
    private:
        string full_name;
};

int main() {
    Student kevin;
    kevin.set_full_name("Kevin Lee");
}
```

Setters

If you have both getters and setters, why not just make the variable public?

```
class Student {
    public:
        ...
        string get_full_name() {
            return full_name;
        }
        void set_full_name(string name) {
            full_name = name;
        }
    private:
        string full_name;
};

int main() {
    Student kevin;
    kevin.set_full_name("Kevin Lee");
}
```

Intermission

A program that uses lots of classes and instances is called *object-oriented*

Carving raccoons out of a tree
https://www.youtube.com/embed/2a1QISYNGHs?rel=0

Change your words - Change your world
https://www.youtube.com/watch?v=Hzgzim5m7oU

Compiling Classes

- We can also separate the declaration of a class from its definition
  - This is another use of **.h** header files

Compiling Classes

In Student.h:

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

class Student {
    public:
        Student(string first_name,
                string last_name);
        string get_full_name();
        void set_full_name(string name);
        void print_name();
    private:
        string full_name;
};
```

Compiling Classes

In `Student.cpp`:

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

Student::Student(string first_name,
                 string last_name) {
    full_name = first_name + " " + last_name;
}
string Student::get_full_name() {
    return full_name;
}
void Student::set_full_name(string name) {
    full_name = name;
}
void Student::print_name() {
    cout << full_name << endl;
}
```

Compiling Classes

In `Student.cpp`:

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

Student::Student(string first_name,
                 string last_name) {
    full_name = first_name + " " + last_name;
}
string Student::get_full_name() {
    return full_name;
}
void Student::set_full_name(string name) {
    full_name = name;
}
void Student::print_name() {
    cout << full_name << endl;
}
```

The `Student::` tells C++ that the function is part of the `Student` class.

Compiling Classes

In `Student.cpp`:

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

Student::Student(string first_name,
                 string last_name) {
    full_name = first_name + " " + last_name;
}
string Student::get_full_name() {
    return full_name;
}
void Student::set_full_name(string name) {
    full_name = name;
}
void Student::print_name() {
    cout << full_name << endl;
}
```

Student:: known as Scope Resolution operator

The `Student::` tells C++ that the function is part of the `Student` class.

Compiling Classes

In `main.cpp`:

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

int main() {
    Student helen("Helen", "Hagos");
    helen.print_name();
    cout << helen.get_full_name() << endl;
}
```

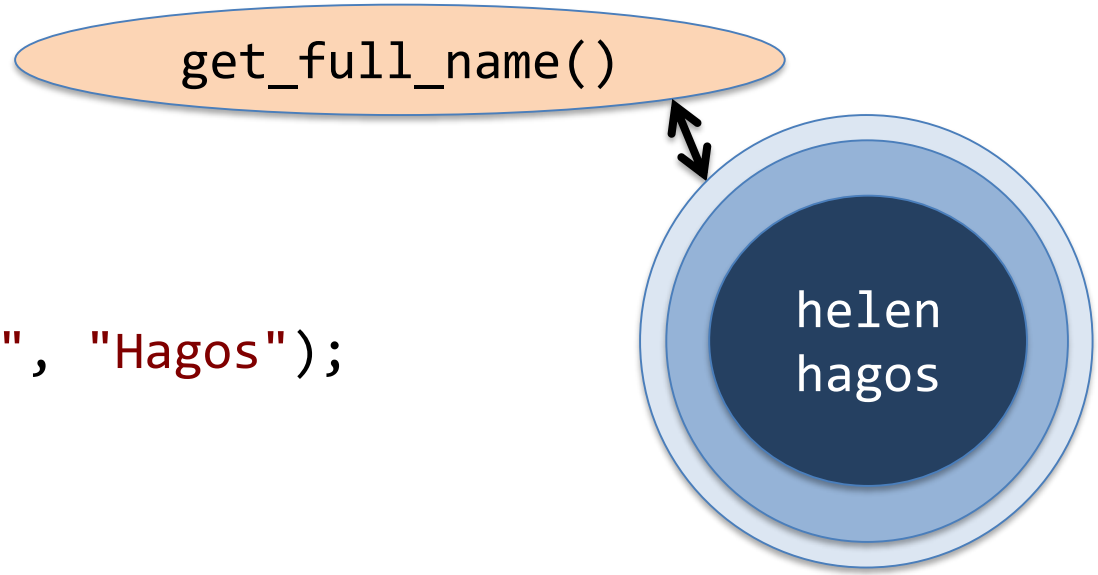
Console

```
Helen Hagos
Helen Hagos
```

Visualization

In `main.cpp`:

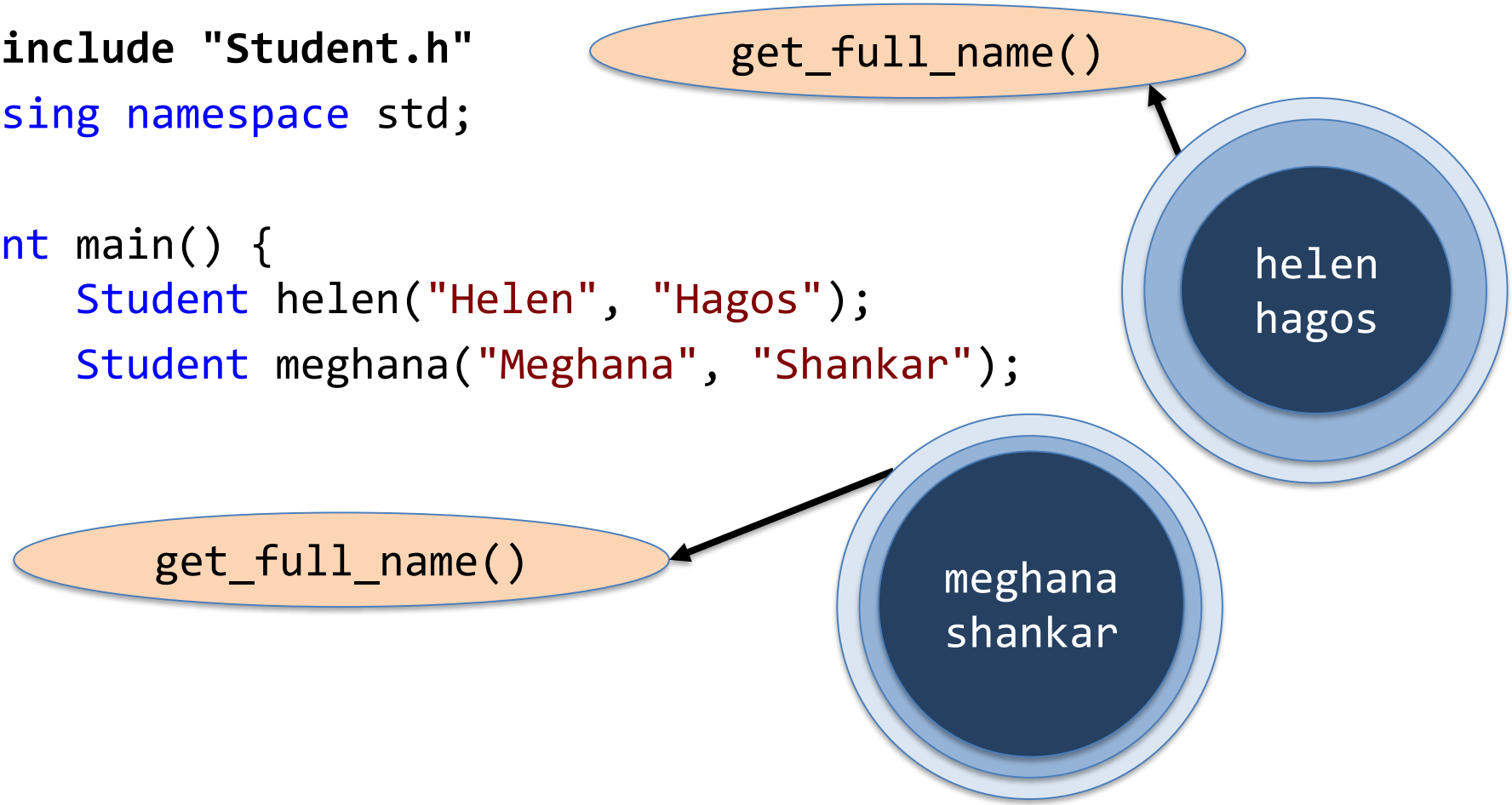
```
#include <string>
#include <iostream>
#include "Student.h"
using namespace std;

int main() {
    Student helen("Helen", "Hagos");
}
```



Visualization

In `main.cpp`:

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

int main() {
    Student helen("Helen", "Hagos");
    Student meghana("Meghana", "Shankar");
}
```



The Card Class

```
const char DIAMONDS = 'D';
const char CLUBS = 'C';
const char HEARTS = 'H';
const char SPADES = 'S';

class Card {
    public:
        char suit;
        int rank;
};
```

i>clicker #7

```
const char DIAMONDS = 'D';
const char CLUBS = 'C';
const char HEARTS = 'H';
const char SPADES = 'S';

class Card {
    public:
        char suit;
        int rank;
};
```

Given this class definition, which of these would correctly initialize a Card?

A) `Card c = {SPADES, 1};`
B) `Card c(SPADES, 1);`
C)
```
Card c;
c.suit = SPADES;
c.rank = 1;
```
D) A and C
E) All of the above

i>clicker #8

```
const char DIAMONDS = 'D';
const char CLUBS = 'C';
const char HEARTS = 'H';
const char SPADES = 'S';

class Card {
    private:
        char suit;
        int rank;
};
```

Given this class definition, which of these would correctly initialize a Card?

A) `Card c = {SPADES, 1};`
B) `Card c(SPADES, 1);`
C)
```
Card c;
c.suit = SPADES;
c.rank = 1;
```
D) A and C
E) None of the above

i>clicker #9

```
class Card {
    public:
        char getSuit() {
            ?????
        }
    private:
        char suit;
        int rank;
};
```

What should be the body of the *getter* function?

A) `return suit;`

B) `int suit = SPADES;`
`return suit;`

C) `return SPADES;`

D) None of the above

i>clicker #10

```
// in Card.h
class Card {
    public:
        Card(char inSuit, int inRank);
    private:
        char suit;
        int rank;
};
```

What is
`Card(char inSuit, int inRank);`
called?

A) a function
B) a constructor
C) a mess

Review: Classes in Multiple Files

How do we write the constructor for Card?

```
// in Card.cpp
#include "Card.h"

// The Constructor for Card
Card::Card(char inSuit, int inRank) {
    suit = inSuit;
    rank = inRank;
}
```

Zyante Review: Default Constructors

- A *default constructor* is a constructor without any arguments

```
Card::Card() {
    suit = SPADES;
    rank = 1;
}
```

Zyante Review: Default Constructors

- We can have multiple constructors, a technique known as ***overloading***

```
Card::Card() {
    suit = SPADES;
    rank = 1;
}
Card::Card(char inSuit, int inRank) {
    suit = inSuit;
    rank = inRank;
}
```

Zyante Review: Default Constructors

- The compiler will intelligently pick which function to use based on the arguments

```
Card::Card() {
    suit = SPADES;
    rank = 1;
}
Card::Card(char inSuit, int inRank) {
    suit = inSuit;
    rank = inRank;
}
```

Zyante Review: Default Constructors

- The compiler will intelligently pick which function to use based on the arguments

```
// calls default constructor
Card card1;
// calls non-default constructor
Card card2(HEART, 12);
```

Zyante Review: Default Constructors

- The compiler will intelligently pick which function to use based on the arguments

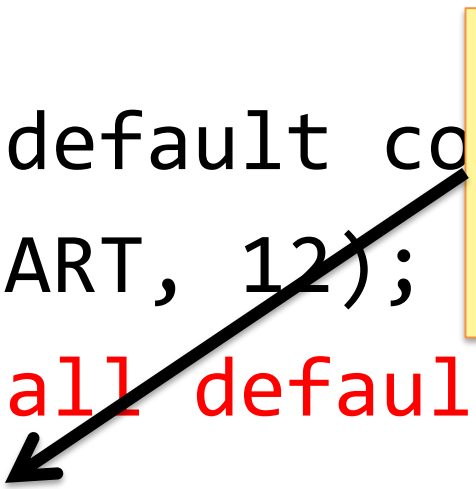
```
// calls default constru
Card card1;
// calls non-default cor
Card card2(HEART, 12);
```

Leaving out the parenthesis is the way of calling the default constructor.

Zyante Review: Default Constructors

- The compiler will intelligently pick which function to use based on the arguments

```
// calls default constructor
Card card1;
// calls non-default co
Card card2(HEART, 12);
// DOES NOT call default constructor
Card card1();
```



Compiler *may* give a warning. This is a function declaration!

iClicker #11: Default Constructors

```
Card::Card(char inSuit, int inRank) {
    suit = inSuit;
    rank = inRank;
}

Card::Card() {
    suit = SPADES;
    rank = 1;
}
```

Which constructor gets called?

```
Card card3(SPADES, 1);
```

A) Default constructor

B) Non-Default constructor

C) Both

D) Neither (compiler error)

iClicker #11: Default Constructors

```
Card::Card(char inSuit, int inRank) {
    suit = inSuit;
    rank = inRank;
}

Card::Card() {
    suit = SPADES;
    rank = 1;
}
```

Which constructor gets called?

```
Card card3(SPADES, 1);
```

A) Default constructor

B) Non-Default constructor

C) Both

D) Neither (compiler error)

Zyante Review: Default Constructors

- An empty default constructor means member variables have default values

```
class Card {
    public:
        Card();
        Card(char inSuit, int inRank);
    private:
        char suit = DIAMONDS; // 'D'
        int rank = 0;
};
```

```
Card::Card() {
}
```

```
int main() {
    Card c;
}
```

- The Card c will be a Diamond

iClicker: Empty Default Constructors

```
class Card {
    public:
        Card();
        Card(char inSuit, int inRank);
    private:
        char suit = DIAMONDS;
        int rank = 0;
};
```

```
Card::Card() {
}
```

```
int main() {
    Card c;
}
```

What is the rank of the Card c?

A) 1 (Ace)
B) 13 (King)
C) 0 (???)
D) None of the above

iClicker: Empty Default Constructors

```
class Card {
    public:
        Card();
        Card(char inSuit, int inRank);
    private:
        char suit = DIAMONDS;
        int rank = 0;
};
```

```
Card::Card() {
}
```

```
int main() {
    Card c;
}
```

What is the rank of the Card c?

A) 1 (Ace)
B) 13 (King)
C) 0 (???)
D) None of the above

iClicker: Empty Default Constructors

```
class Card {
    public:
        Card();
        Card(char inSuit, int inRank);
    private:
        char suit;
        int rank;
};
```

```
Card::Card() {
   suit = CLUBS;
   rank = 2;
}
```

```
int main() {
    Card c;
}
```

setting default values in declaration are a C++ 11 feature
won't work in previous versions, so setting in the default constructor is better.

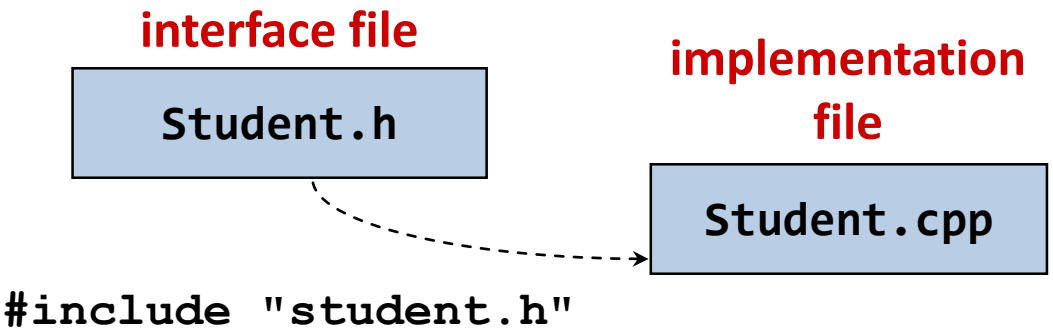
Default Constructors Summary

- A constructor with no arguments is called a default constructor
- We can have multiple constructors as long as they have different arguments (through function **overloading**)
- An empty default constructor means variables have their default values

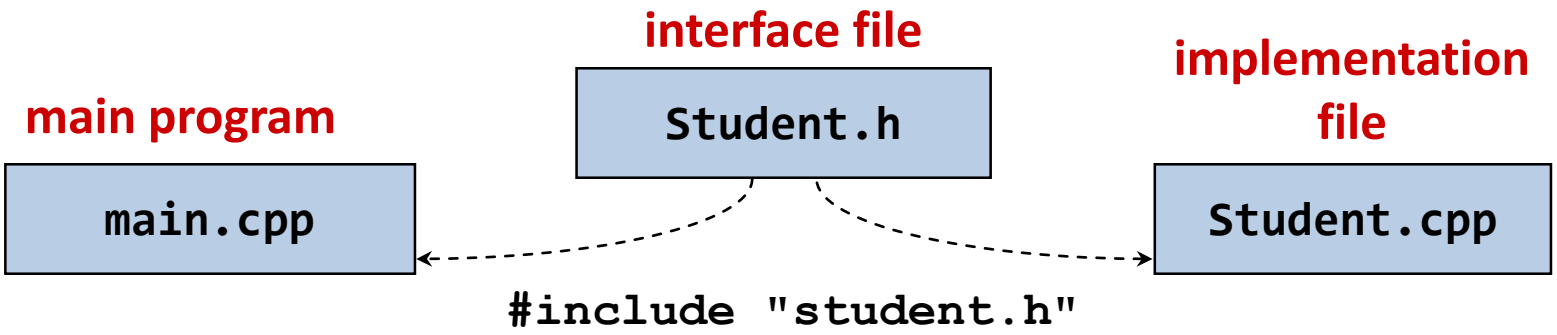
Separate Compilation and Linking of Files

interface file

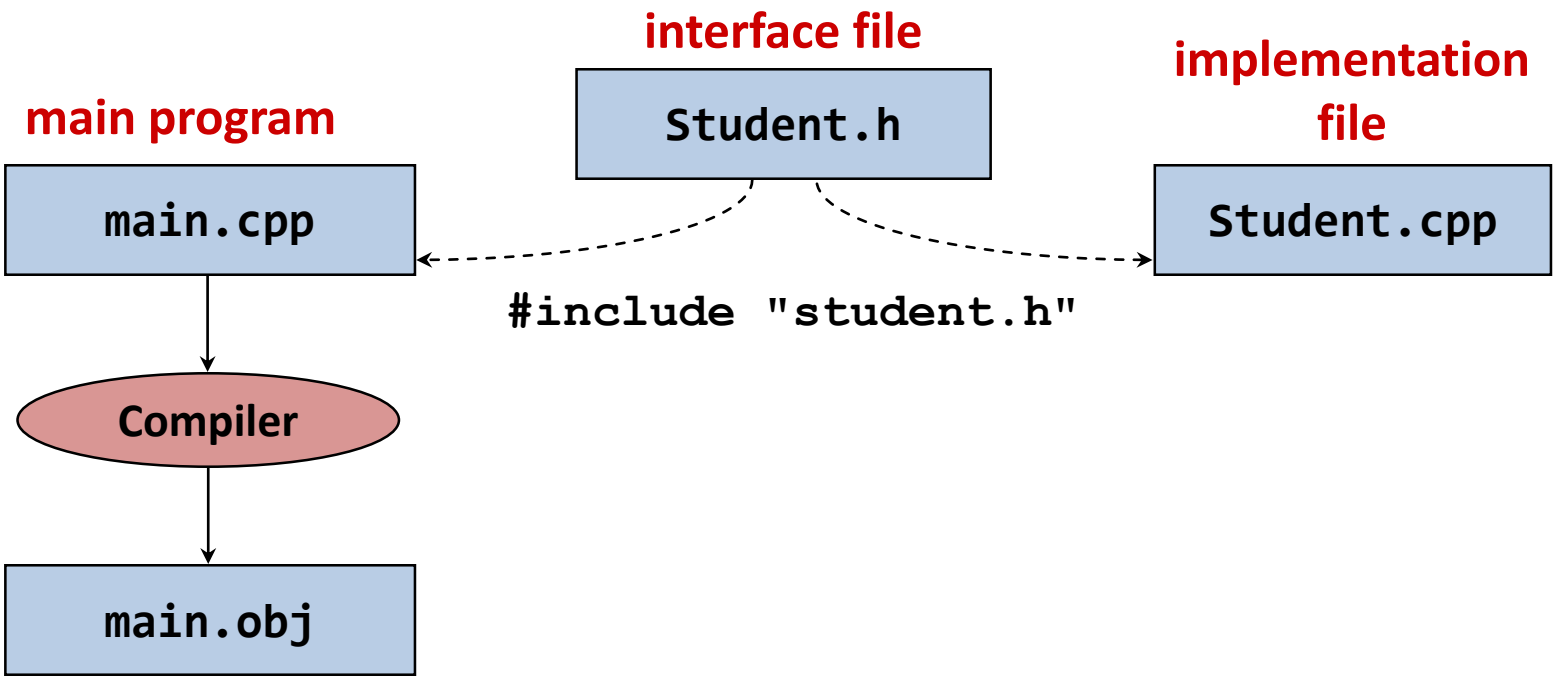
Separate Compilation and Linking of Files



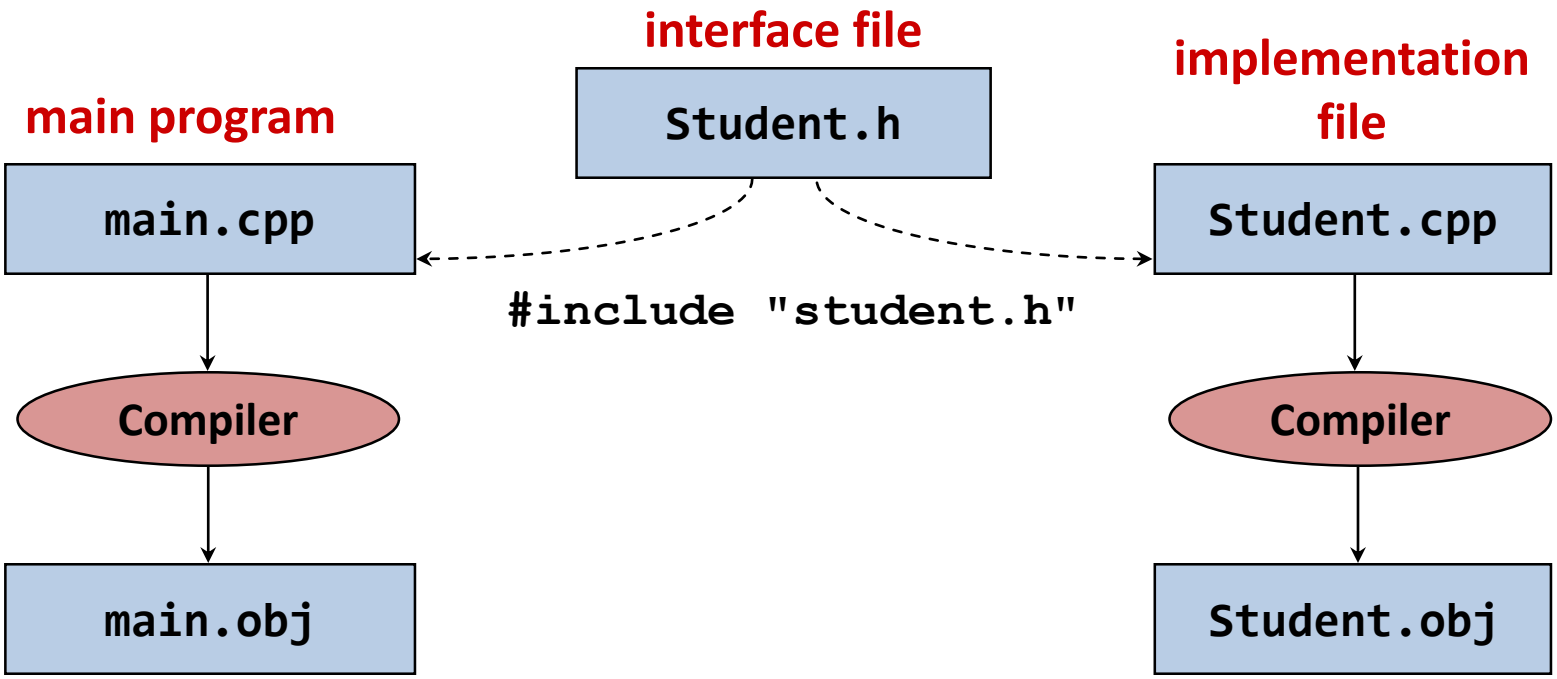
Separate Compilation and Linking of Files



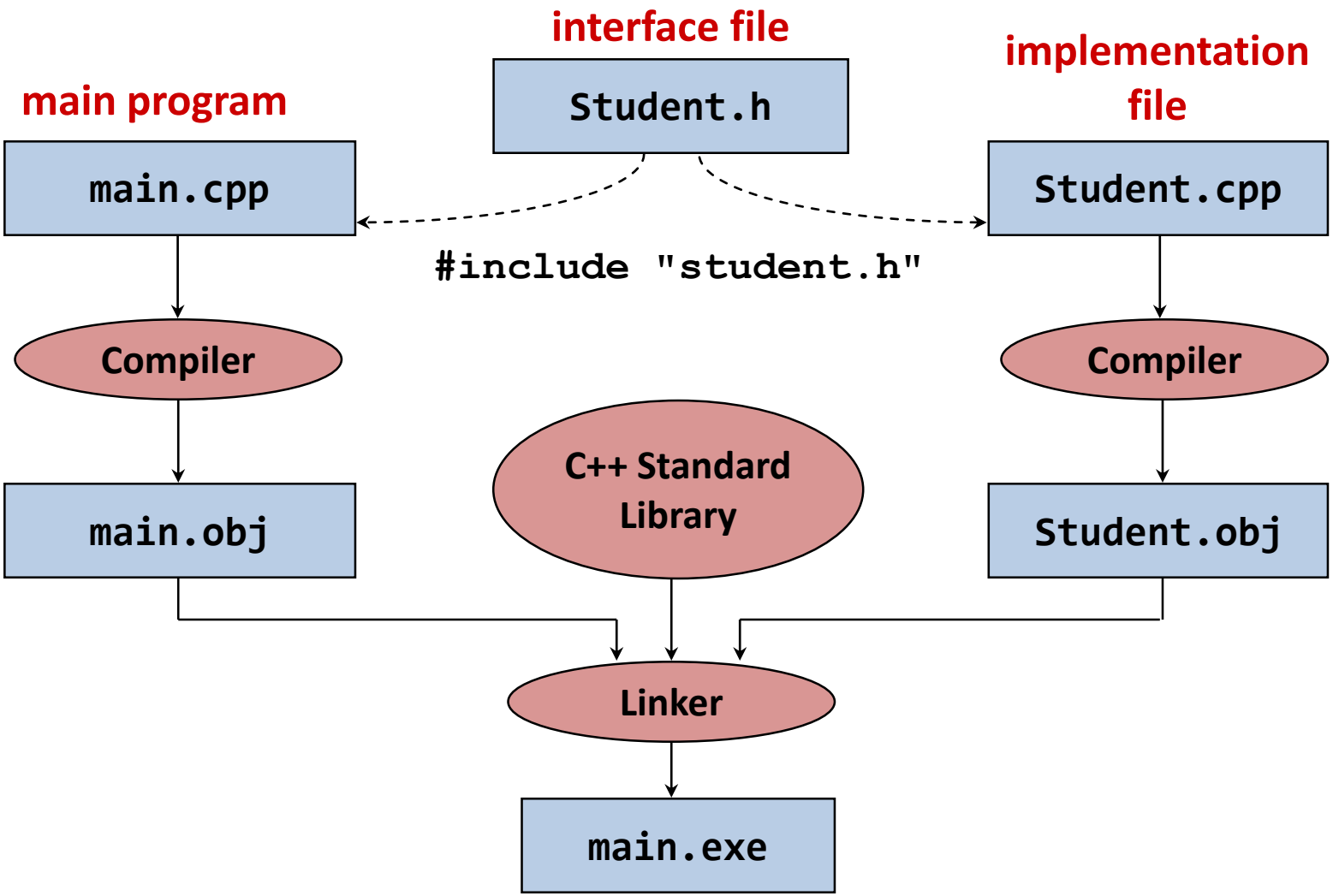
Separate Compilation and Linking of Files



Separate Compilation and Linking of Files



Separate Compilation and Linking of Files



Summary

- Classes have **public** and **private** members
- For private members, we need **constructors** to initialize an instance of the class
- We can then use **getters** and **setters** to change that data.
- We can separate a class into:
  - **declarations** (in the .h file) and
  - **definitions** (in the .cpp file)

Next Class: More Classes!

Member Functions

(Classes never end…)