22B. Constructors, Live Coding with Classes

CPSC 120: Introduction to Programming
Pratishtha Soni~ CSU Fullerton

Agenda

1. Constructors
2. Live Coding with Classes

1. Constructors

Review: Google-Style Class Declaration (.h file)

declaration:

class *identifier* **{**
public:
member-function-declaration...

private:
data-member-declaration...
};

Semantics:

- Creates *identifier* as a class data type
- Class has data members and member functions that are declared

```
class WordFrequency {
public:
 WordFrequency(const std::string& word,
               int frequency);

 const std::string& Word() const;
 int Frequency() const;

private:
 std::string word_;
 int frequency_;
};
```

Review: Constructors

- **Constructor:** special member function that initializes an object when it is created
- Expressions between { } become arguments to constructor

Sequence of events:

1. initializers
2. body of constructor

```
class WordFrequency {
public:
 WordFrequency(const std::string& word,
               int frequency);

 const std::string& Word() const;
 int Frequency() const;

private:
 std::string word_;
 int frequency_;
};
```

Review: Each Object Has Its Own Data Members

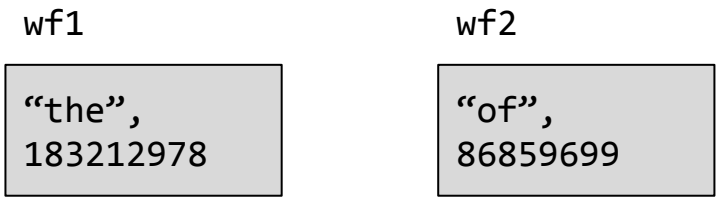
```
int main(int argc, char* argv[]) {
 WordFrequency wf1{"the", 183212978};
 WordFrequency wf2{"of", 86859699};

 std::cout << wf1.Word() << " " << wf1.Frequency() << "\n";
 std::cout << wf2.Word() << " " << wf2.Frequency() << "\n";

 return 0;
}
```

Output:

```
the 183212978
of 86859699
```



Review: Data Members are in Scope

- Data members are in scope in member functions
- The purpose of member functions is to manipulate data members

```
const std::string& WordFrequency::Word()
const {
 return word_;
}

int WordFrequency::Frequency() const {
 return frequency_;
}
```

Review: Syntax: Constructor Definition (.cc file)

constructor-definition:

class::*class*(*parameter...*) : *initializer...* {
body-statement... }

(same as regular function definition, plus *class*::, minus return type)

Semantics:

- The constructor is defined

```
WordFrequency::WordFrequency(
    const std::string& word,
    int frequency)
: word_(word), frequency_(frequency) { }
```

Review: Syntax: Initializer (.cc file)

initializer:

member-variable(*expresssion*)

Semantics:

- *member-variable* is initialized to *expression*
- Happens **before** the body of the constructor

```
WordFrequency::WordFrequency(
    const std::string& word,
    int frequency)
: word_(word), frequency_(frequency) { }
```

Default Constructor

- **Default constructor:** constructor with no arguments
- “Default” because no arguments are given when object is created
- Needs to initialize all data members **without** any arguments
- Omitted from classes where this is impossible
- Required to store class in vector

Example: Video Game Scoreboard

Example: Default Constructor

```
// .h file

class ScoreboardEntry {
public:
 ScoreboardEntry();

 const std::string& Name() const;
 int Score() const;

private:
 std::string name_;
 int score_;
};
```

```
// .cc file

ScoreboardEntry::ScoreboardEntry()
: name_(""), score_(0) { }

const std::string& ScoreboardEntry::Name() const {
 return name_;
}

int ScoreboardEntry::Score() const {
 return score_;
}
```

Example: Calling Default Constructor

```
// .h file

class ScoreboardEntry {
public:
 ScoreboardEntry();

 const std::string& Name() const;
 int Score() const;

private:
 std::string name_;
 int score_;
};
```

```
// main.cc

int main(int argc, char* argv[]) {
 ScoreboardEntry entry1;

 return 0;
}
```

Overloading

- **Overload** (v): multiple versions of the same function
- Same name
- parameter types and/or return type differ
    - at least one thing must be different so the versions can be told apart
- **Deduce** (v): arrive at a conclusion by reasoning
- When function is called, compiler **deduces** which version to call based on data types

Overloaded Constructor

- **Overloaded constructor:** when class constructors are overloaded
- Multiple constructors
- Parameter types are different for each
  - Recall: constructor has no return type
- When object is created, compiler deduces which constructor to use from the arguments
- Examples
  - `std::vector::vector`
  - `std::string::string`

Example: Overloaded Constructor

```
// .h file

class ScoreboardEntry {
public:
 ScoreboardEntry();

 ScoreboardEntry(const std::string& name,
                 int score);

 const std::string& Name() const;
 int Score() const;

private:
 std::string name_;
 int score_;
};
```

```
// .cc file

ScoreboardEntry::ScoreboardEntry()
: score_(0) { }

ScoreboardEntry::ScoreboardEntry(
  const std::string& name, int score)
: name_(name), score_(score) { }

const std::string& ScoreboardEntry::Name() const {
 return name_;
}

int ScoreboardEntry::Score() const {
 return score_;
}
```

Example: Calling Overloaded Constructor

```
// .h file

class ScoreboardEntry {
public:
 ScoreboardEntry();

 ScoreboardEntry(const std::string& name,
                 int score);

 const std::string& Name() const;
 int Score() const;

private:
 std::string name_;
 int score_;
};
```

```
// main.cc

int main(int argc, char* argv[]) {
 ScoreboardEntry entry1;
 ScoreboardEntry entry2{"DRJ", 21270};
 ScoreboardEntry entry3{"SAM", 18315};
 ScoreboardEntry entry4;

 return 0;
}
```

2. Live Coding with Classes

Review: Live Coding

- Interactive
- Instructor: **driver**
- Students: **navigators**

Prompts

1. Create a class for a **restaurant menu item**
    a. Name, price, vegetarian, spicy
2. Define constructors, accessors, mutators
3. Store in vector