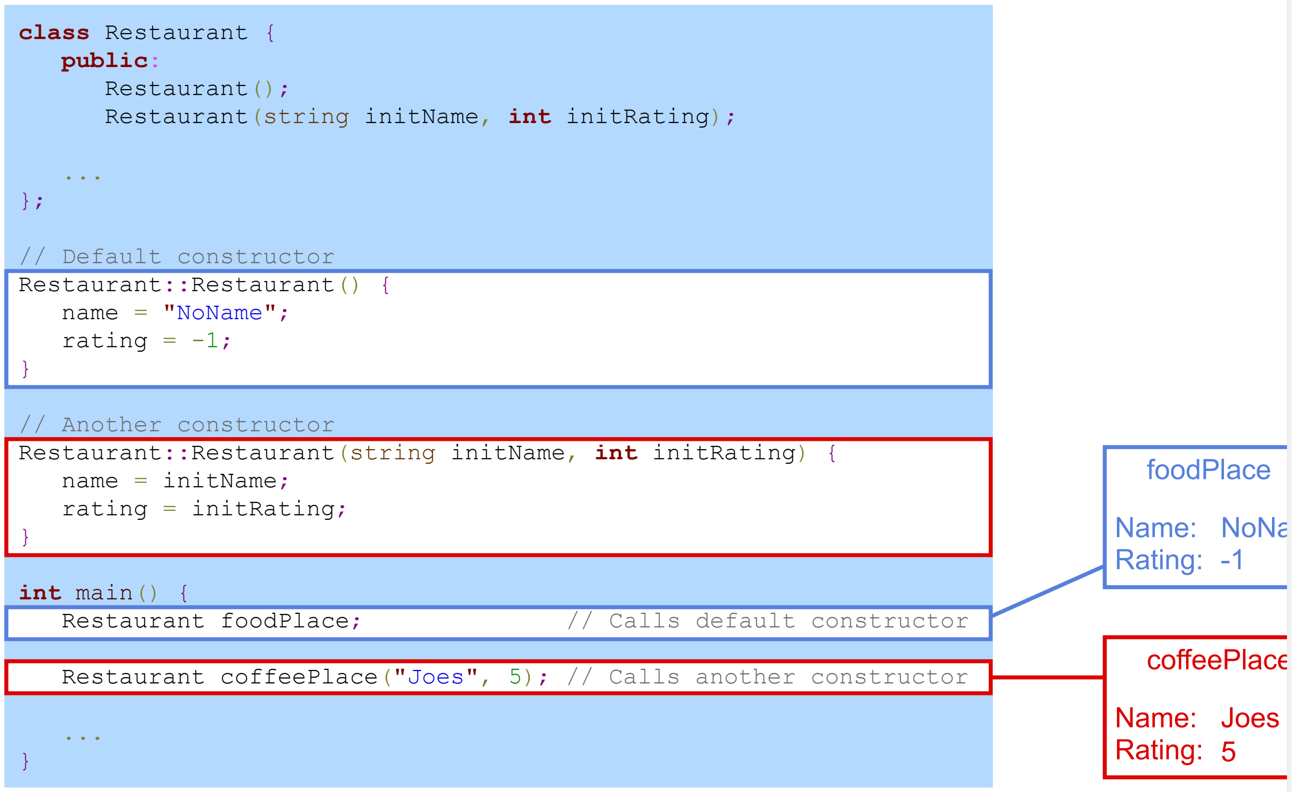
7.11 Constructor overloading

Basics

Programmers often want to provide different initialization values when creating a new object. A class creator can ***overload*** a constructor by defining multiple constructors differing in parameter types. A variable declaration can have arguments. The constructor with matching parameters will be called.

PARTICIPATION ACTIVITY 7.11.1: Overloaded constructors.

Start 2x speed



```
class Restaurant {
   public:
      Restaurant();
      Restaurant(string initName, int initRating);

   ...
};

// Default constructor
Restaurant::Restaurant() {
   name = "NoName";
   rating = -1;
}

// Another constructor
Restaurant::Restaurant(string initName, int initRating) {
   name = initName;
   rating = initRating;
}

int main() {
   Restaurant foodPlace;                  // Calls default constructor

   Restaurant coffeePlace("Joes", 5); // Calls another constructor

   ...
}
```

Feedback?

zyDE 7.11.1: Overloading a constructor.

Load default template...

Run

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

class Restaurant {
   public:
      Restaurant();
      Restaurant(string initName, int initR
      void Print();

    private:
      string name;
      int rating;
};

// Default constructor
Restaurant::Restaurant() {
   name = "NoName";
   rating = -1;
}
```

Feedback?

PARTICIPATION ACTIVITY

7.11.2: Overloaded constructors.

Given the three constructors below, indicate which will be called for each declaration.

```
class SomeClass {
   SomeClass();                         // A
   SomeClass(string name);              // B
   SomeClass(string name, int num);     // C
}
```

1) `SomeClass myObj("Lee");`

A [✓ B] C Error

Correct

The one string argument matches the one string parameter of B.

2) `SomeClass myObj();`

A B [✓ Error]

Correct

Although calling a regular function with no arguments requires parentheses, for a variable declaration the parentheses must be omitted, else a compiler error occurs.

3) `SomeClass myObj;`

- [✓ A]
- B
- Error

Correct

The declaration has no argument and constructor A has no parameter, so constructor A is called. A constructor called when no arguments exist is the default constructor.

4) `SomeClass myObj("Lee", 5, 0);`

- C
- [✓ Error]

Correct

The declaration has three arguments, but no constructor has three parameters. The compiler can't find a match so generates an error.

5) `vector<SomeClass> myVect(5);`

- [✓ A]
- Error

Correct

myVect is of type vector, so the 5 is used for the vector's constructor. That constructor will create 5 objects of type SomeClass. Lacking arguments, those will use the default SomeClass() constructor.

Feedback?

If any constructor defined, should define default

If a programmer defines any constructor, the compiler does not implicitly define a default constructor, so good practice is for the programmer to also explicitly define a default constructor so that a declaration like `MyClass x;` remains supported.

Figure 7.11.1: The programmer defined a constructor, so the compiler does not automatically define a default constructor.

```
class Restaurant {
   public:
      Restaurant(string initName, int
initRating);

      // No other constructors
      ...
};

int main() {
   Restaurant foodPlace;
   ...
}
```

```
tmp1.cpp:37:15: error: no matching constructor for
initialization of
        'Restaurant'
    Restaurant foodPlace;
```

Feedback?

PARTICIPATION ACTIVITY | 7.11.3: Constructor definitions. ✓

Which of the following is OK as the entire set of constructors for class MyClass? Assume a declaration like `MyClass x;` should be supported.

1) `MyClass();`

[✓ OK] Error

Correct

The programmer is defining the default constructor. ✓

2) `// None`

[✓ OK] Error

Correct

If the programmer doesn't define any constructors, the compiler implicitly defines a default constructor having no statements. ✓

3)
```
MyClass();
MyClass(string name);
```

[✓ OK] Error

Correct

The programmer has defined the default constructor (doesn't require any arguments), and another constructor that requires one argument. ✓

4) `MyClass(string name);`

OK [✓ Error]

Correct

If the programmer defines any constructor, then the programmer should define a default constructor. Failing to do so yields a compiler error for a declaration like `MyClass x;`. ✓

Feedback?

Constructors with default parameter values

Like any function, a constructor's parameters may be assigned default values.

If those default values allow the constructor to be called without arguments, then that constructor can serve as the default constructor.

The default values could be in the function definition, but are clearer to class users in the declaration.

Figure 7.11.2: A constructor with default parameter values can serve as the default constructor.

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

class Restaurant {
   public:
      Restaurant(string initName = "NoName", int initRating = -1);
      void Print();

    private:
      string name;
      int rating;
};

Restaurant::Restaurant(string initName, int initRating) {
   name = initName;
   rating = initRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant foodPlace;
   Restaurant coffeePlace("Joes", 5);

   foodPlace.Print();
   coffeePlace.Print();

   return 0;
}
```

```
NoName -- -1
Joes -- 5
```

Feedback?

PARTICIPATION ACTIVITY | 7.11.4: Constructor with default parameter values may serve as default constructor. ✓

Which of the following is OK as the entire set of constructors for class YourClass? Assume a declaration like `YourClass obj;` should be supported.

1) `YourClass();`

[✓ OK] Error

Correct

The programmer is defining the default constructor. ✓

2) `YourClass(string name, int num);`

OK [✓ Error]

Correct

If a programmer defines any constructor, the programmer should also define a default constructor. Here, the default constructor is missing. Thus, a declaration like `YourClass obj;` will yield a compiler error. ✓

3)
```
YourClass(string name =
"", int num = 0);
```

[✓ OK] Error

Correct

Due to the default parameter values, the constructor supports these declarations:

```
// Initializes with "" and 0
YourClass obj;

// Initializes with "Mary" and 0
YourClass obj("Mary");

// Initializes with "Joe" and 1
YourClass obj("Joe", 1);
```

Because the constructor can be called without arguments, the constructor serves as a default constructor.

4)
```
YourClass();
YourClass(string name =
"", int num = 0);
```

OK [✓ Error]

Correct

Both constructors can be called with no arguments, so the compiler doesn't know which to call as the default constructor. A compiler error will occur, due to the ambiguity.

Feedback?

CHALLENGE ACTIVITY 7.11.1: Enter the output of the constructor overloading.

Jump to level 1

Type the program's output.

```
Luna, -
Rio, 5
Unnamed
```

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

class Pet {
   public:
      Pet(string petName = "Unnamed", int yearsOld = -1);
      void Print();

    private:
      string name;
      int age;
};

Pet::Pet(string petName, int yearsOld) {
   name = petName;
   age = yearsOld;
}

void Pet::Print() {
   cout << name << ", " << age << endl;
}

int main() {
   Pet dog;
   Pet cat("Luna");
   Pet bird("Rio", 5);

   cat.Print();
   bird.Print();
   dog.Print();

   return 0;
}
```

1 2 3

Check Next **Done**. Click any level to practice more. Completion is preserv

✓ Each Pet declaration calls the constructor. `Pet dog` calls the constructor without passing a parameter values are used. `Pet cat("Luna")`passes "Luna" to the first constructor parameter the default parameter value. `Pet bird("Rio", 5)` passes arguments to both parameters.

Yours

```
Luna, -1
Rio, 5
Unnamed, -1
```

Expected

```
Luna, -1
Rio, 5
Unnamed, -1
```

Feedback?

CHALLENGE ACTIVITY 7.11.2: Constructor overloading.

Write a second constructor as indicated. Sample output:

```
User1: Minutes: 0, Messages: 0
User2: Minutes: 1000, Messages: 5000
```

```
}

// FIXME: Create a second constructor with numMinutes and numMessages parameters.

/* Your solution goes here  */
PhonePlan::PhonePlan(int inputfreeMinutes, int inutfreeMessages) {     // Default constru
   freeMinutes  = inputfreeMinutes;
   freeMessages = inutfreeMessages;
}

void PhonePlan::Print() const {
   cout << "Minutes: " << freeMinutes << ", Messages: " << freeMessages << endl;
}

int main() {
   PhonePlan user1Plan;                 // Calls default constructor
   PhonePlan user2Plan(1000, 5000);     // Calls newly-created constructor

   cout << "User1: ";
   user1Plan.Print();

   cout << "User2: ";
```

Run ✓ All tests passed

✓ Checking default constructor.

Your output

```
User1: Minutes: 0, Messages: 0
```

✓ Checking overloaded constructor.

Your output

```
User2: Minutes: 1000, Messages: 5000
```

Feedback?