7.12 Constructor initializer lists

A **constructor initializer list** is an alternative approach for initializing data members in a constructor, coming after a colon and consisting of a comma-separated list of variableName(initValue) items.

Figure 7.12.1: Member initialization: (left) Using statements in the constructor, (right) Using a constructor initializer list.

```
#include <iostream>
                                            #include <iostream>
using namespace std;
                                           using namespace std;
class SampleClass {
                                            class SampleClass {
   public:
                                               public:
      SampleClass();
                                                  SampleClass();
      void Print() const;
                                                  void Print() const;
   private:
                                               private:
      int field1;
                                                  int field1;
      int field2;
                                                  int field2;
};
                                           };
SampleClass::SampleClass() {
                                           SampleClass::SampleClass() : field1(100),
   field1 = 100;
                                            field2(200) {
   field2 = 200;
void SampleClass::Print() const {
   cout << "Field1: " << field1 <<</pre>
                                            void SampleClass::Print() const {
end1;
                                              cout << "Field1: " << field1 << endl;</pre>
   cout << "Field2: " << field2 <<</pre>
                                              cout << "Field2: " << field2 << endl;</pre>
end1;
                                           int main() {
int main() {
                                               SampleClass myClass;
   SampleClass myClass;
                                               myClass.Print();
   myClass.Print();
                                               return 0;
   return 0;
Field1: 100
                                            Field1: 100
Field2: 200
                                            Field2: 200
```

Feedback?

PARTICIPATION ACTIVITY

7.12.1: Member initialization.

Answer

 Convert this constructor to use a constructor initializer list.

```
MyClass::MyClass() {
    x = -1;
    y = 0;
}

MyClass::MyClass()

{
}

Check Show answer
```

: x(-1), y(0)

The colon indicates that initializations follow, consisting here of a commaseparated list of pairs of the form: variableName(initValue).

Feedback?

The approach is important when a data member is a class type that must be explicitly constructed. Otherwise, that data member is by default constructed. Ex: If you have studied vectors, consider a data member consisting of a vector of size 2.

Figure 7.12.2: Member initialization in a constructor.

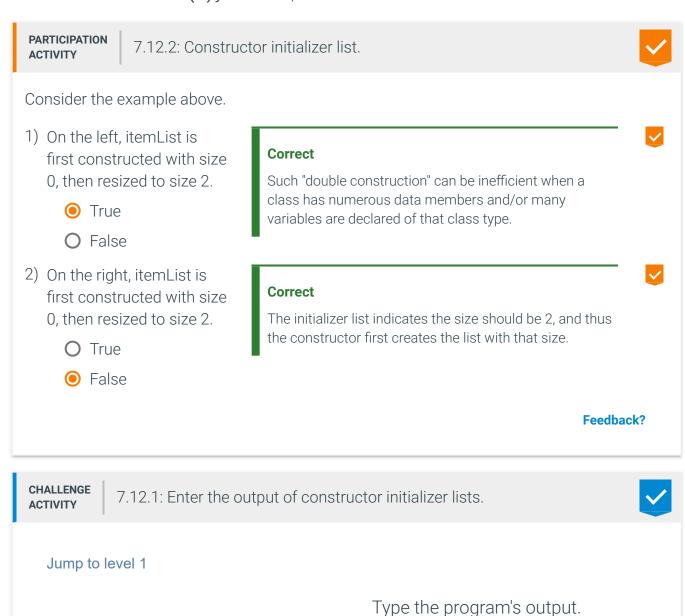
```
#include <iostream>
#include <vector>
using namespace std;
class SampleClass {
   public
      SampleClass();
      void Print() const;
   private:
      vector<int> itemList;
};
SampleClass::SampleClass() {
  // itemList gets default constructed,
size 0
   itemList.resize(2);
void SampleClass::Print() const {
   cout << "Item1: " << itemList.at(0) <<</pre>
end1:
   cout << "Item2: " << itemList.at(1) <<</pre>
end1;
int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
```

```
#include <iostream>
#include <vector>
using namespace std;
class SampleClass {
   public:
      SampleClass();
      void Print() const;
   private:
      vector<int> itemList;
SampleClass::SampleClass() : itemList(2) {
   // itemList gets constructed with size 2
void SampleClass::Print() const {
   cout << "Item1: " << itemList.at(0) <<</pre>
   cout << "Item2: " << itemList.at(1) <<</pre>
end1;
int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```



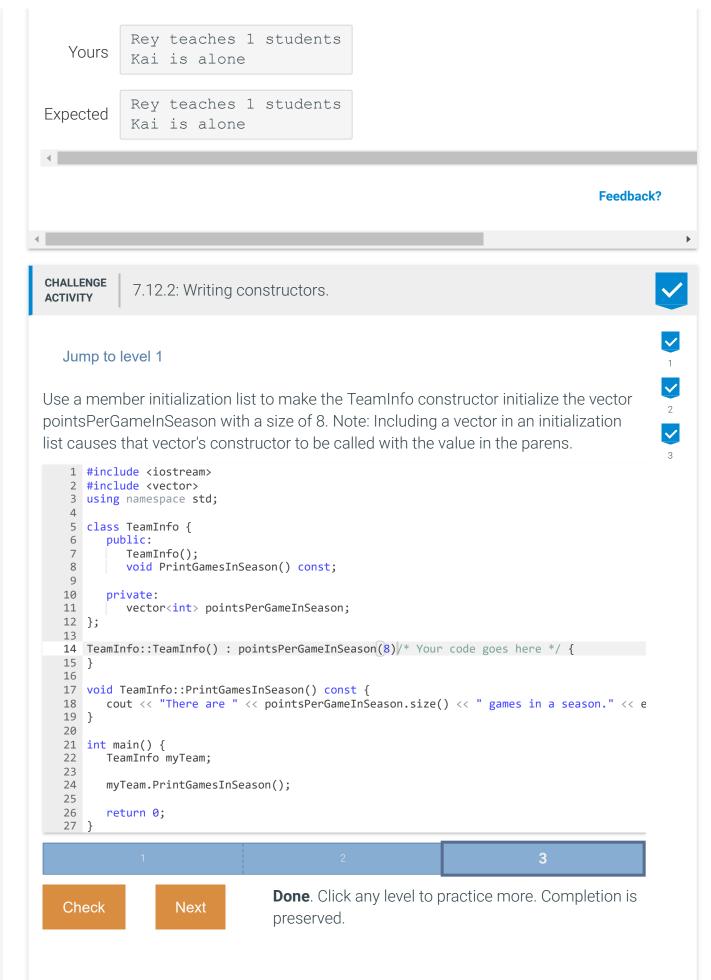
On the left, the constructor initially creates a vector of size 0, then resizes to size 2. On the right, itemList(2) is provided in the SampleClass constructor initialization list, causing the vector constructor to be called with size 2. Using the initialization list avoids the inefficiency of constructing and then modifying an item.

Note: Since C++11, the data member could have been initialized in the class definition: vector<int> itemList(2);. However, initialization lists are still useful for other cases.



```
#include <iostream>
#include <string>
using namespace std;
class Students {
   public:
      Students();
      Students (int);
      int GetTotal() const { return numStudents; }
   private:
      int numStudents;
};
Students::Students() : numStudents(0) {}
Students::Students(int num) : numStudents(num) {}
class Tutor {
   public:
      Tutor(string);
      Tutor(string, int);
      void Print() const;
   private:
      string name;
      Students students;
};
Tutor::Tutor(string tutorName) : name(tutorName) {}
Tutor::Tutor(string tutorName, int numStudents) : name(tutorName), students(numStudents)
void Tutor::Print() const {
   if (students.GetTotal() == 0){
      cout << name << " is alone" << endl;</pre>
   else {
      cout << name << " teaches " << students.GetTotal() << " students" << endl;</pre>
int main() {
   Tutor myTutor("Kai");
   Tutor yourTutor("Rey", 1);
   yourTutor.Print();
   myTutor.Print();
   return 0;
  Check
                                 Done. Click any level to practice more. Completion is preserv
```

✓ A different constructor is called for the Student variable depending on which Tutor construct tutorName) does not assign the private data member students with any parameter, so the defa whereas Tutor(string tutorName, int numStudents) assigns students with a number, therefore called.



There are 8 games in a season.

✓ If instead, the size was 9, then write listOfPointsInSeason(9) to create a vector of size 9. However, the instructions say to initialize with a size of 8.

✓ 1: Compare output ∧

Feedback?

CHALLENGE ACTIVITY

7.12.3: Creating a constructor with a constructor initializer list.



Complete the PoundDog code by adding a constructor having a constructor initializer list that initializes age with 1, id with -1, and name with "NoName". Notice that MyString's default constructor does *not* get called.

Note: If you instead create a traditional default constructor as below, MyString's default constructor will be called, which prints output and thus causes this activity's test to fail. Try it!

```
// A wrong solution to this activity...
PoundDog::PoundDog() {
   age = 1;
   id = -1;
   name.SetString("NoName");
}
```

Your output

```
24 class PoundDog {
25
       public:
26
          PoundDog();
          void Print() const;
27
28
29
       private:
30
          int age;
31
          int id;
32
          MyString name;
33 };
34
35 /* Your solution goes here */
36
37 PoundDog::PoundDog() : age(1), id(-1), name("NoName"){}
38
39 void PoundDog::Print() const {
       cout << "age: " << age << endl;
cout << "id: " << id << endl;</pre>
40
41
       cout << "name: " << name.GetString() << endl;</pre>
42
43 }
44
45 int main() {
```

Run

All tests passed

✓ Testing

Your output

id: -1
name: NoName

Feedback?

Exploring further:

- Classes from cplusplus.com, see "Member initialization in constructors" section.
- Constructors from msdn.microsoft.com, see "Member lists".