

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>
using namespace std;

class SampleClass {
public:
    SampleClass();
    void Print() const;

private:
    int field1;
    int field2;
};

SampleClass::SampleClass() {
    field1 = 100;
    field2 = 200;
}

void SampleClass::Print() const {
    cout << "Field1: " << field1 << endl;
    cout << "Field2: " << field2 << endl;
}

int main() {
    SampleClass myClass;
    myClass.Print();
    return 0;
}
```

```
Field1: 100
Field2: 200
```

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

class SampleClass {
public:
    SampleClass();
    void Print() const;

private:
    int field1;
    int field2;
};

SampleClass::SampleClass() : field1(100),
field2(200) {
}

void SampleClass::Print() const {
    cout << "Field1: " << field1 << endl;
    cout << "Field2: " << field2 << endl;
}

int main() {
    SampleClass myClass;
    myClass.Print();
    return 0;
}
```

```
Field1: 100
Field2: 200
```

[Feedback?](#)

- 1) Convert this constructor to use a constructor initializer list.

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

```
MyClass::MyClass() 
{
}
```

Check

Show answer

Answer

```
: x(-1), y(0)
```

The colon indicates that initializations follow, consisting here of a comma-separated 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) <<
endl;
    cout << "Item2: " << itemList.at(1) <<
endl;
}

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) <<
endl;
    cout << "Item2: " << itemList.at(1) <<
endl;
}

int main() {
    SampleClass myClass;
    myClass.Print();
    return 0;
}
```

Item1: 0
Item2: 0

Item1: 0
Item2: 0

[Feedback?](#)

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.

**PARTICIPATION
ACTIVITY**

7.12.2: Constructor initializer list.



Consider the example above.

- 1) On the left, `itemList` is first constructed with size 0, then resized to size 2.

☒ True
☐ False

Correct

Such "double construction" can be inefficient when a class has numerous data members and/or many variables are declared of that class type.



- 2) On the right, `itemList` is first constructed with size 0, then resized to size 2.

☐ True
☒ False

Correct

The initializer list indicates the size should be 2, and thus the constructor first creates the list with that size.

[Feedback?](#)**CHALLENGE
ACTIVITY**

7.12.1: Enter the output of constructor initializer lists.

[Jump to level 1](#)

Type the program's output.

```

#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;
    }
    else {
        cout << name << " teaches " << students.GetTotal() << " students" << endl;
    }
}

int main() {
    Tutor myTutor("Kai");
    Tutor yourTutor("Rey", 1);

    yourTutor.Print();
    myTutor.Print();

    return 0;
}

```

1

2

3

4

Check

Next

Done. Click any level to practice more. Completion is preserv

A different constructor is called for the Student variable depending on which Tutor construc
 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.

Yours

```
Rey teaches 1 students  
Kai is alone
```

Expected

```
Rey teaches 1 students  
Kai is alone
```

[Feedback?](#)**CHALLENGE
ACTIVITY**

7.12.2: Writing constructors.

[Jump to level 1](#)

Use a member initialization list to make the TeamInfo constructor initialize the vector pointsPerGameInSeason with a size of 8. Note: Including a vector in an initialization list causes that vector's constructor to be called with the value in the parens.



1



2



3

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 class TeamInfo {
6     public:
7         TeamInfo();
8         void PrintGamesInSeason() const;
9
10    private:
11        vector<int> pointsPerGameInSeason;
12 };
13
14 TeamInfo::TeamInfo() : pointsPerGameInSeason(8)/* Your code goes here */ {
15 }
16
17 void TeamInfo::PrintGamesInSeason() const {
18     cout << "There are " << pointsPerGameInSeason.size() << " games in a season." << endl;
19 }
20
21 int main() {
22     TeamInfo myTeam;
23
24     myTeam.PrintGamesInSeason();
25
26     return 0;
27 }
```

1

2

3

Check

Next

Done. Click any level to practice more. Completion is preserved.

✓ 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 ^

Your output

There are 8 games in a season.

[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");
}
```

```
24 class PoundDog {
25     public:
26         PoundDog();
27         void Print() const;
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 {
40     cout << "age: " << age << endl;
41     cout << "id: " << id << endl;
42     cout << "name: " << name.GetString() << endl;
43 }
44
45 int main() {
```

Run

✓ All tests passed

✓ Testing

Your output

```
age: 1
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".