

15.x — Chapter 15 summary and quiz

👤 ALEX¹ ⌚ MAY 3, 2024

Chapter Review

Inside every (non-static) member function, the keyword **this** is a const pointer that holds the address of the current implicit object. We can have functions return `*this` by reference in order to enable **method chaining**, where several member functions can be called on the same object in a single expression.

Prefer to put your class definitions in a header file with the same name as the class. Trivial member functions (such as access functions, constructors with empty bodies, etc...) can be defined inside the class definition.

Prefer to define non-trivial member functions in a source file with the same name as the class.

A type that is defined inside a class type is called a **nested type** (or **member type**). Type aliases can also be nested.

Member functions defined inside a class template definition can use the template parameters of the class template itself. Member functions defined outside the class template definition must resupply a template parameter declaration, and should be defined (in the same file) just below the class template definition.

Static member variables are static duration members that are shared by all objects of the class. Static members exist even if no objects of the class have been instantiated. Prefer to access them using the class name, the scope resolution operator, and the members name.

Making static members `inline` allows them to be initialized inside the class definition.

Static member functions are member functions that can be called with no object. They do not have a `*this` pointer, and cannot access non-static data members.

Inside the body of a class, a **friend declaration** (using the `friend` keyword) can be used to tell the compiler that some other class or function is now a friend. A **friend** is a class or function (member or non-member) that has been granted full access to the private and protected members of another class. A **friend function** is a function (member or non-member) that can access the private and protected members of a class as though it were a member of that class. A **friend class** is a class that can access the private and protected members of another class.

Quiz time

Question #1

Let's create a random monster generator. This one should be fun.

a) First, let's create an scoped enumeration of monster types named `MonsterType`. Include the following monster types: Dragon, Goblin, Ogre, Orc, Skeleton, Troll, Vampire, and Zombie. Add an additional `maxMonsterTypes` enumerator so we can count how many enumerators there are.

[Show Solution](#) (javascript:void(0))²

b) Now, let's create our `Monster` class. Our `Monster` will have 4 attributes (member variables): a type (`MonsterType`), a name (`std::string`), a roar (`std::string`) and the number of hit points (`int`).

[Show Solution](#) (javascript:void(0))²

c) `enum class MonsterType` is specific to `Monster`, so make `MonsterType` a nested unscoped enum inside `Monster` and rename it to `Type`.

[Show Solution](#) (javascript:void(0))²

d) Create a constructor that allows you to initialize all of the member variables.

The following program should compile:

```
1 | int main()
2 | {
3 |     Monster skeleton{ Monster::skeleton, "Bones", "*rattle*", 4 };
4 |
5 |     return 0;
6 | }
```

[Show Solution](#) (javascript:void(0))²

e) Now we want to be able to print our monster so we can validate it's correct. Write two functions: One called `getTypeString()` that returns the monster's type as a string, and one called `print()` that matches the output in the sample program below.

The following program should compile:

```
1 | int main()
2 | {
3 |     Monster skeleton{ Monster::skeleton, "Bones", "*rattle*", 4 };
4 |     skeleton.print();
5 |
6 |     Monster vampire{ Monster::vampire, "Nibblez", "*hiss*", 0 };
7 |     vampire.print();
8 |
9 |     return 0;
10 | }
```

and print:

```
Bones the skeleton has 4 hit points and says *rattle*.
Nibblez the vampire is dead.
```

[Show Solution](#) (javascript:void(0))²

f) Now we can create a random monster generator. Let's consider how our `MonsterGenerator` will work. Ideally, we'll ask it to give us a `Monster`, and it will create a random one for us. Because `MonsterGenerator` doesn't have any state, this is a good candidate for a namespace.

Create a `MonsterGenerator` namespace. Create function within named `generate()`. This should return a `Monster`. For now, make it return `Monster{ Monster::skeleton, "Bones", "*rattle*", 4 }`;

The following program should compile:

```
1 | int main()
2 | {
3 |     Monster m{ MonsterGenerator::generate() };
4 |     m.print();
5 |
6 |     return 0;
7 | }
```

and print:

```
Bones the skeleton has 4 hit points and says *rattle*
```

[Show Solution](#) (javascript:void(0))²

g) Add two more functions to the `MonsterGenerator` namespace. `getName(int)` will take a number between 0 and 5 (inclusive) and return a name of your choice. `getRoar(int)` will also take a number between 0 and 5 (inclusive) and return a roar of your choice. Also update your `generate()` function to call `getName(0)` and `getRoar(0)`.

The following program should compile:

```
1 | int main()
2 | {
3 |     Monster m{ MonsterGenerator::generate() };
4 |     m.print();
5 |
6 |     return 0;
7 | }
```

and print:

```
Blarg the skeleton has 4 hit points and says *ROAR*
```

Your name and sound will vary based on what you chose.

[Show Solution](#) (javascript:void(0))²

h) Now we'll randomize our generated monster. Grab the "Random.h" code from [8.15 -- Global random numbers \(Random.h\)](#) (<https://www.learncpp.com/cpp-tutorial/global-random-numbers-random-h/#RandomH>)³ and save it as `Random.h`. Then use `Random::get()` to generate a random monster type, random name, random roar, and random hit points (between 1 and 100).

The following program should compile:

```
1 #include "Random.h"
2
3 int main()
4 {
5     Monster m{ MonsterGenerator::generate() };
6     m.print();
7
8     return 0;
9 }
```

and print something like this:

Mort the zombie has 61 hit points and says *growl*


[Show Solution](#) (javascript:void(0))²



Next lesson

16.1 [Introduction to containers and arrays](#)

4



[Back to table of contents](#)

5



Previous lesson

15.10 [Ref qualifiers](#)

6

7



B U URL INLINE CODE C++ CODE BLOCK HELP!


Leave a comment...

 Name*

 Email*  

 Find a mistake? Leave a comment above!

 Avatars from <https://gravatar.com/>⁹ are connected to your provided email address.

Notify me about replies: 

POST COMMENT



Peter the goblin
🕒 June 24, 2025 7:48 am PDT

Peter the goblin has 12 hit points and says "[[It Burns! Ow! Stop! Help Me! It Burns!]]".

👍 1 ➡ Reply



Copernicus
🕒 June 8, 2025 7:30 am PDT

Question #1


```

1  #include <iostream>
2  #include <random>
3  #include <string>
4  #include <string_view>
5
6  //Random generator
7  namespace Random
8  {
9      std::random_device rd{};
10     std::seed_seq ss{ rd(), rd(), rd(), rd(), rd(), rd(), rd(), rd() };
11     std::mt19937 mt{ ss };
12
13     int generateRandomNumber(int startRange, int endRange)
14     {
15         std::uniform_int_distribution range{ startRange, endRange };
16         return range(mt);
17     }
18 }
19
20 //Monster implementation
21 class Monster
22 {
23 public:
24     enum Race
25     {
26         error,
27         dragon,
28         goblin,
29         orge,
30         orc,
31         skeleton,
32         troll,
33         vampire,
34         zombie,
35         maxRaceTypes,
36     };
37 public:
38     Monster(Race race, std::string name, std::string roar, int hitPoints)
39         : m_race{ race }, m_name{ name }, m_roar{ roar }, m_hitPoints{ hitPoints }
40     {
41     }
42     ~Monster() = default;
43
44     std::string getName() const
45     {
46         return m_name;
47     }
48     std::string getRoar() const
49     {
50         return m_roar;
51     }
52     int getHitPoints() const
53     {
54         return m_hitPoints;
55     }
56     std::string getRace() const
57     {
58         switch (m_race)
59         {
60             case Monster::error:
61                 return "???";
62             case Monster::dragon:
63                 return "Dragon";
64             case Monster::goblin:
65                 return "Goblin";
66             case Monster::orge:
67                 return "Orge";
68             case Monster::orc:
69                 return "Orc";
70             case Monster::skeleton:

```

```

71         return "Skeleton";
72     case Monster::troll:
73         return "Troll";
74     case Monster::vampire:
75         return "Vampire";
76     case Monster::zombie:
77         return "Zombie";
78     case Monster::maxRaceTypes:
79         return "???";
80     default:
81         return "Unknown";
82     }
83 }
84
85 private:
86     Race m_race{};
87     std::string m_name{ "???" };
88     std::string m_roar{ "???" };
89     int m_hitPoints{ 100 };
90 };
91
92 static const void displayMonster(Monster& monster)
93 {
94     std::cout << monster.getName() << " the " << monster.getRace() << " has " <<
monster.getHitPoints() << " hit points and say's " << monster.getRoar() << '\n';
95 }
96
97 //Generate a random monster
98 static const Monster::Race generateRace()
99 {
100     switch (Random::generateRandomNumber(1, 8))
101     {
102     case 1:
103         return Monster::Race::dragon;
104     case 2:
105         return Monster::Race::goblin;
106     case 3:
107         return Monster::Race::orge;
108     case 4:
109         return Monster::Race::orc;
110     case 5:
111         return Monster::Race::skeleton;
112     case 6:
113         return Monster::Race::troll;
114     case 7:
115         return Monster::Race::vampire;
116     case 8:
117         return Monster::Race::zombie;
118     default:
119         return Monster::Race::error;
120     }
121 }
122
123 static const std::string generateRandomName()
124 {
125     switch (Random::generateRandomNumber(1, 10))
126     {
127     case 1:
128         return "Kraken";
129     case 2:
130         return "Bigfoot";
131     case 3:
132         return "Hydra";
133     case 4:
134         return "Bleeding Hollowman";
135     case 5:
136         return "Dreadfen Wyrn";
137     case 6:
138         return "Skuldrake";
139     case 7:
140         return "Gannic Sentinel";

```



```

140         return "Carriion Sentinel";
141     case 8:
142         return "Velthrax";
143     case 9:
144         return "Ulgarok";
145     case 10:
146         return "Molgrith";
147     default:
148         return "Vermin";
149     }
150 }
151
152 static const std::string generateRoar()
153 {
154     switch (Random::generateRandomNumber(1, 10))
155     {
156     case 1:
157         return "Grrraahhh-rrrahhh!";
158     case 2:
159         return "RROOOAAARRR!";
160     case 3:
161         return "YII-HAH-RAAAHH!";
162     case 4:
163         return "GRAAAH-HAHHH!";
164     case 5:
165         return "RAAAHHRRRRGHHH!";
166     case 6:
167         return "SHHHRRR-RAAAA!";
168     case 7:
169         return "GRRROOAAAARRRR!";
170     case 8:
171         return "RRAAAAGGGHHHHH!";
172     case 9:
173         return "SSS-KHHRRAAAAA!";
174     case 10:
175         return "WOOOOOAAAARRRRGHHH!";
176     default:
177         return "S-SSSSSSS-RAAAH!";
178     }
179 }
180
181 static const Monster generateRandomMonster()
182 {
183     return { generateRace(), generateRandomName(), generateRoar(),
184             Random::generateRandomNumber(1, 100) };
185 }
186
187 int main()
188 {
189     Monster test{ generateRandomMonster() };
190     displayMonster(test);
191
192     return 0;
193 }

```



```

1  #include <iostream>
2  #include <string>
3  #include <random>
4
5  // Simple Random utility namespace
6  namespace Random
7  {
8      std::random_device rd; // Non-deterministic random device
9      std::mt19937 mt{ rd() }; // Mersenne Twister engine seeded with rd()
10
11     // Generate random int between min and max (inclusive)
12     int get(int min, int max)
13     {
14         std::uniform_int_distribution<> dist(min, max);
15         return dist(mt);
16     }
17 }
18
19 // Monster class
20 class Monster
21 {
22 public:
23     // Nested unscoped enum Type for monster types
24     enum Type
25     {
26         dragon,
27         goblin,
28         ogre,
29         orc,
30         skeleton,
31         troll,
32         vampire,
33         zombie,
34         maxMonsterTypes
35     };
36
37 private:
38     Type type;
39     std::string name;
40     std::string roar;
41     int hitPoints;
42
43 public:
44     // Constructor
45     Monster(Type type, std::string name, std::string roar, int hitPoints)
46         : type(type), name(std::move(name)), roar(std::move(roar)),
47         hitPoints(hitPoints) {}
48
49     // Return monster type as string
50     std::string getTypeString() const
51     {
52         switch (type)
53         {
54             case dragon:    return "dragon";
55             case goblin:    return "goblin";
56             case ogre:      return "ogre";
57             case orc:       return "orc";
58             case skeleton:  return "skeleton";
59             case troll:     return "troll";
60             case vampire:   return "vampire";
61             case zombie:    return "zombie";
62             default:        return "unknown";
63         }
64     }
65
66     // Print monster info
67     void print() const
68     {
69         if (hitPoints > 0)
70         {

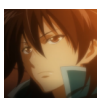
```

```

71     " hit points and says " << roar << ".\n";
72     }
73     else
74     {
75         std::cout << name << " the " << getTypeString() << " is dead.\n";
76     }
77 }
78 };
79
80 // MonsterGenerator namespace
81 namespace MonsterGenerator
82 {
83     // Preset names
84     std::string getName(int index)
85     {
86         static const std::string names[] = { "Blarg", "Mort", "Glim", "Nim", "Zed",
87 "Rex" };
88         if (index < 0 || index >= 6)
89             return "Unknown";
90         return names[index];
91     }
92
93     // Preset roars
94     std::string getRoar(int index)
95     {
96         static const std::string roars[] = { "*ROAR*", "*growl*", "*hiss*",
97 "*rattle*", "*grunt*", "*snarl*" };
98         if (index < 0 || index >= 6)
99             return "*silence*";
100        return roars[index];
101    }
102
103    // Generate a random monster
104    Monster generate()
105    {
106        Monster::Type type = static_cast<Monster::Type>(Random::get(0,
107 Monster::maxMonsterTypes - 1));
108        std::string name = getName(Random::get(0, 5));
109        std::string roar = getRoar(Random::get(0, 5));
110        int hitPoints = Random::get(1, 100);
111
112        return Monster(type, name, roar, hitPoints);
113    }
114 }
115
116 int main()
117 {
118     Monster m = MonsterGenerator::generate();
119     m.print();
120
121     return 0;
122 }

```

👍 0 ➡ Reply



Zeca

🕒 May 3, 2025 8:02 am PDT

took me one hour but there is it


```

1  #include <iostream>
2  #include "Random.h"
3
4  class Monster
5  {
6  public:
7      enum Type
8      {
9          dragon,
10         goblin,
11         ogre,
12         orc,
13         skeleton,
14         troll,
15         vampire,
16         zombie,
17         maxMonsterTypes
18     };
19
20 private:
21     Type m_type{};
22     std::string m_name{};
23     std::string m_roar{};
24     int m_hp{};
25
26 public:
27     Monster(Type type, std::string_view name, std::string_view roar, const int hp)
28         : m_type{type}, m_name{name}, m_roar{roar}, m_hp{hp}
29     {
30
31     }
32
33     std::string_view getTypeString()
34     {
35         switch(m_type)
36         {
37             case dragon:
38                 return "dragon";
39             case goblin:
40                 return "goblin";
41             case ogre:
42                 return "ogre";
43             case orc:
44                 return "orc";
45             case skeleton:
46                 return "skeleton";
47             case troll:
48                 return "troll";
49             case vampire:
50                 return "vampire";
51             case zombie:
52                 return "zombie";
53             default:
54                 return "???";
55         }
56     }
57
58     void print(void)
59     {
60         if(m_hp == 0)
61         {
62             std::cout << m_name << " the " << getTypeString() << " is dead.\n";
63         }
64         else
65             std::cout << m_name << " the " << getTypeString() << " has " << m_hp << "
hit points and says " << m_roar << ".\n";
66     }
67 };
68
69 namespace MonsterGenerator

```

```

70 {
71     std::string_view getName(int a)
72     {
73         switch(a)
74         {
75             case 0:
76                 return "Blarg";
77             case 1:
78                 return "Bones";
79             case 2:
80                 return "Nibblez";
81             case 3:
82                 return "Dragulon";
83             case 4:
84                 return "Zarneth";
85             case 5:
86                 return "Thirzagul";
87             default:
88                 return "???";
89         }
90     }
91
92     std::string_view getRoar(int a)
93     {
94         switch(a)
95         {
96             case 0:
97                 return "*ROAR*";
98             case 1:
99                 return "*SKREEEEEEAAARRGH*";
100             case 2:
101                 return "*GRAAAAWRRRRHH*";
102             case 3:
103                 return "*KRAZAK'THUL*";
104             case 4:
105                 return "*VROOOORNNN*";
106             case 5:
107                 return "*ZHRRRAAKKH*";
108             default:
109                 return "???";
110         }
111     }
112
113     Monster generate()
114     {
115         return {static_cast<Monster::Type>(Random::get(0,7)),
116             getName(Random::get(0,5)), getRoar(Random::get(0,5)), Random::get(1, 100)};
117     }
118
119     int main()
120     {
121         Monster m{ MonsterGenerator::generate() };
122         m.print();
123
124         return 0;
125     }

```

👍 0 ➡ Reply



Aayush

🕒 May 1, 2025 11:35 am PDT

Phew its finally working


```

1 #include <iostream>
2 #include <string>
3 #include "Random.h"
4
5 class Monster
6 {
7 public:
8     enum Type
9     {
10         dragon,
11         goblin,
12         ogre,
13         orc,
14         skeleton,
15         troll,
16         vampire,
17         zombie,
18         maxMonsterTypes,
19     };
20
21 private:
22     Type m_type{};
23     std::string m_name{"???"};
24     std::string m_roar{"???"};
25     int m_hitPoints{};
26
27 public:
28     Monster(Type type, std::string_view name, std::string_view roar, int hitPoints)
29         : m_type{type}, m_name{name}, m_roar{roar}, m_hitPoints{hitPoints}
30     {
31     }
32
33     constexpr std::string GetTypeString() const
34     {
35         switch (m_type)
36         {
37             case dragon: return "dragon";
38             case goblin: return "goblin";
39             case ogre: return "ogre";
40             case orc: return "orc";
41             case skeleton: return "skeleton";
42             case troll: return "troll";
43             case vampire: return "vampire";
44             case zombie: return "zombie";
45             case maxMonsterTypes: return "8";
46         }
47
48         return "???";
49     }
50
51     void print()
52     {
53         if (m_hitPoints <= 0)
54         {
55             std::cout << m_name << " the " << GetTypeString() << " is " << "dead " <<
56             "\n";
57         }
58         else
59         {
60             std::cout << m_name << " the " << GetTypeString() << " has " <<
61             m_hitPoints << " hit points and says " << m_roar << "\n";
62         }
63     };
64
65     namespace MonsterGenerator
66     {
67
68         std::string_view getName(int name)
69         {

```

```

70     switch(name)
71     {
72         case 0: return "Blarg";
73         case 1: return "wushang";
74         case 2: return "lonewolf";
75         case 3: return "gourav";
76         case 4: return "aayush";
77         case 5: return "adi";
78         default: return "???";
79     }
80 }
81
82 std::string_view getRoar(int roar)
83 {
84     switch(roar)
85     {
86         case 0: return "*ROAR*";
87         case 1: return "nihao";
88         case 2: return "aauuuu";
89         case 3: return "shota";
90         case 4: return "hentai";
91         case 5: return "racist";
92         default: return "???";
93     }
94 }
95
96 Monster generate()
97 {
98     return Monster{static_cast<Monster::Type>(Random::get(0, 7)),
99     getName(Random::get(0, 5)), getRoar(Random::get(0, 5)), Random::get(1, 100)};
100 }
101
102 int main()
103 {
104     Monster m{ MonsterGenerator::generate() };
105     m.print();
106
107     return 0;
108 }

```

 Last edited 2 months ago by Aayush

 0  Reply



Aayush

🕒 May 1, 2025 10:32 am PDT

i think i might have forgotten something or maybe i just didn't pay attention but for some reason i don't understand why

```

1  #include <iostream>
2  #include <string>
3
4  class Monster
5  {
6  public:
7      enum Type
8      {
9          Dragon,
10         Goblin,
11         Ogre,
12         Orc,
13         Skeleton,
14         Troll,
15         Vampire,
16         Zombie,
17         MaxMonsterTypes,
18     };
19
20 private:
21     Type m_type{};
22     std::string m_name{"???"};
23     std::string m_roar{"???"};
24     int m_hitPoints{};
25
26 public:
27     Monster (Type type, std::string& name, std::string& roar, int hitPoints)
28         : m_type{type}
29         , m_name{name}
30         , m_roar{roar}
31         , m_hitPoints{hitPoints}
32     {
33     }
34 };
35
36 int main()
37 {
38     Monster skeleton { Monster::Skeleton, "Bones", "*rattle*", 4 };
39
40     return 0;
41 }

```

this does not work, like we cannot use reference there?

👍 0 ➡ Reply



Dinny

🕒 March 15, 2025 7:55 pm PDT

Prolapse the vampire has 79 hitpoints and says *squelch*.

👍 1 ➡ Reply



Aiden

🕒 March 10, 2025 3:47 am PDT

Monster.h and Monster.cpp

```
1  #ifndef MONSTER_H
2  #define MONSTER_H
3
4  #include <string>
5
6  class Monster
7  {
8  public:
9      enum Type
10     {
11         dragon,
12         goblin,
13         orc,
14         troll,
15         ogre,
16         skeleton,
17         vampire,
18         zombie,
19         maxMonsterTypes
20     };
21
22     Monster(Type type, std::string_view name, std::string roar, int hitPoint)
23         : m_type{ type }, m_name{ name }, m_roar{ roar }, m_hitPoint{ hitPoint }
24     {
25     }
26
27     std::string getTypeString(Type type) const;
28
29     void print() const;
30 private:
31     Type m_type{};
32     std::string m_name{"???"};
33     std::string m_roar{"???"};
34     int m_hitPoint{};
35 };
36
37
38 #endif MONSTER_H
```

```

1  #include "Monster.h"
2  #include <iostream>
3
4
5  std::string Monster::getTypeString(Type type) const
6  {
7      switch (type)
8      {
9          case dragon:
10             return "dragon";
11             break;
12          case goblin:
13             return "goblin";
14             break;
15          case orc:
16             return "orc";
17             break;
18          case troll:
19             return "troll";
20             break;
21          case ogre:
22             return "ogre";
23             break;
24          case skeleton:
25             return "skeleton";
26             break;
27          case vampire:
28             return "vampire";
29             break;
30          case zombie:
31             return "zombie";
32             break;
33          default:
34             return "???";
35             break;
36      }
37  }
38
39  void Monster::print() const
40  {
41      if (m_hitPoint > 0)
42      {
43          std::cout << m_name << " the "
44                  << getTypeString(m_type) << " has "
45                  << m_hitPoint << " hit points and says "
46                  << m_roar << ".\n";
47      }
48      else
49      {
50          std::cout << m_name << " the "
51                  << getTypeString(m_type) << " is dead.\n";
52      }
53  }
54  }

```

main.cpp


```
1 #include <string_view>
2 #include <iostream>
3 #include "Monster.h"
4 #include "Random.h"
5
6 namespace MonsterGenerator
7 {
8     std::string_view getName(int index)
9     {
10         switch (index)
11         {
12             case 0:
13                 return "Blarg";
14                 break;
15             case 1:
16                 return "Moog";
17                 break;
18             case 2:
19                 return "Pksh";
20                 break;
21             case 3:
22                 return "Tyrn";
23                 break;
24             case 4:
25                 return "Mort";
26                 break;
27             case 5:
28                 return "Hans";
29                 break;
30             default:
31                 return "???";
32                 break;
33         }
34     }
35
36     std::string getRoar(int index)
37     {
38         switch (index)
39         {
40             case 0:
41                 return "*RORA*";
42                 break;
43             case 1:
44                 return "*peep*";
45                 break;
46             case 2:
47                 return "*squeal*";
48                 break;
49             case 3:
50                 return "*wine*";
51                 break;
52             case 4:
53                 return "*growl*";
54                 break;
55             case 5:
56                 return "*burp*";
57                 break;
58             default:
59                 return "???";
60                 break;
61         }
62     }
63
64     Monster generate()
65     {
66         int hitPoints = Random::get(1, 100);
67         int name = Random::get(0, 5);
68         int roar = Random::get(0, 5);
69         int type = Random::get(0, 7);
70         return Monster{ static_cast<Monster::Type>(type), getName(name),
```

```

71     getRoar(roar), hitPoints};
72 }
73
74 int main()
75 {
76     Monster m{ MonsterGenerator::generate() };
77     m.print();
78
79     return 0;
80 }

```

👍 0 ➡ Reply



man98

🕒 February 26, 2025 7:04 pm PST

Skelly the skeleton has 46 hit points and says My wife just left me.

👍 8 ➡ Reply



Unai Gonzalez

🕒 January 31, 2025 10:52 am PST

"Grab the "Random.h" code"? no thanks, I'm re-doing the Random Generator myself for practice! :D

👍 2 ➡ Reply



man98

🗨 Reply to [Unai Gonzalez](#) ¹⁰ 🕒 February 22, 2025 4:14 pm PST

<https://youtu.be/JQVBfcjx6d4?t=136>

👍 0 ➡ Reply



Pur1x

🗨 Reply to [Unai Gonzalez](#) ¹⁰ 🕒 February 2, 2025 2:03 am PST

...no thanks!

👍 0 ➡ Reply

Links

1. <https://www.learncpp.com/author/Alex/>
2. `javascript:void(0)`

3. <https://www.learncpp.com/cpp-tutorial/global-random-numbers-random-h/#RandomH>
4. <https://www.learncpp.com/cpp-tutorial/introduction-to-containers-and-arrays/>
5. <https://www.learncpp.com/>
6. <https://www.learncpp.com/cpp-tutorial/ref-qualifiers/>
7. <https://www.learncpp.com/chapter-15-summary-and-quiz/>
8. <https://www.learncpp.com/cpp-tutorial/friend-classes-and-friend-member-functions/>
9. <https://gravatar.com/>
10. <https://www.learncpp.com/cpp-tutorial/chapter-15-summary-and-quiz/#comment-607209>
11. <https://g.ezoic.net/privacy/learncpp.com>