15.x — Chapter 15 summary and quiz

ALEX[1] 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:

```
int main()
{
    Monster skeleton{ Monster::skeleton, "Bones", "*rattle*", 4 };

    return 0;
}
```

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:

```
int main()
{
    Monster skeleton{ Monster::skeleton, "Bones", "*rattle*", 4 };
    skeleton.print();

    Monster vampire{ Monster::vampire, "Nibblez", "*hiss*", 0 };
    vampire.print();

    return 0;
}
```

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:

```
int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

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:

```
int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

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:

```
#include "Random.h"

int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

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

107 COMMENTS

Newest

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

```cpp
#include <iostream>
#include <random>
#include <string>
#include <string_view>

//Random generator
namespace Random
{
    std::random_device rd{};
    std::seed_seq ss{ rd(), rd(), rd(), rd(), rd(), rd(), rd(), rd() };
    std::mt19937 mt{ ss };

    int generateRandomNumber(int startRange, int endRange)
    {
        std::uniform_int_distribution range{ startRange, endRange };
        return range(mt);
    }
}

//Monster implementation
class Monster
{
public:
    enum Race
    {
        error,
        dragon,
        goblin,
        orge,
        orc,
        skeleton,
        troll,
        vampire,
        zombie,
        maxRaceTypes,
    };
public:
    Monster(Race race, std::string name, std::string roar, int hitPoints)
        : m_race{ race }, m_name{ name }, m_roar{ roar }, m_hitPoints{ hitPoints }
    {
    }
    ~Monster() = default;

    std::string getName() const
    {
        return m_name;
    }
    std::string getRoar() const
    {
        return m_roar;
    }
    int getHitPoints() const
    {
        return m_hitPoints;
    }
    std::string getRace() const
    {
        switch (m_race)
        {
        case Monster::error:
            return "???";
        case Monster::dragon:
            return "Dragon";
        case Monster::goblin:
            return "Goblin";
        case Monster::orge:
            return "Orge";
        case Monster::orc:
            return "Orc";
        case Monster::skeleton:
```

```
            return "Skeleton";
        case Monster::troll:
            return "Troll";
        case Monster::vampire:
            return "Vampire";
        case Monster::zombie:
            return "Zombie";
        case Monster::maxRaceTypes:
            return "???";
        default:
            return "Unknown";
        }
    }

private:
    Race m_race{};
    std::string m_name{ "???" };
    std::string m_roar{ "???" };
    int m_hitPoints{ 100 };
};

static const void displayMonster(Monster& monster)
{
    std::cout << monster.getName() << " the " << monster.getRace() << " has " << monster.getHitPoints() << " hit points and say's " << monster.getRoar() << '\n';
}

//Generate a random monster
static const Monster::Race generateRace()
{
    switch (Random::generateRandomNumber(1, 8))
    {
    case 1:
        return Monster::Race::dragon;
    case 2:
        return Monster::Race::goblin;
    case 3:
        return Monster::Race::orge;
    case 4:
        return Monster::Race::orc;
    case 5:
        return Monster::Race::skeleton;
    case 6:
        return Monster::Race::troll;
    case 7:
        return Monster::Race::vampire;
    case 8:
        return Monster::Race::zombie;
    default:
        return Monster::Race::error;
    }
}

static const std::string generateRandomName()
{
    switch (Random::generateRandomNumber(1, 10))
    {
    case 1:
        return "Kraken";
    case 2:
        return "Bigfoot";
    case 3:
        return "Hydra";
    case 4:
        return "Bleeding Hollowman";
    case 5:
        return "Dreadfen Wyrm";
    case 6:
        return "Skuldrake";
    case 7:
        return "Gaspier Sentinel";
```

```
        return "Carrion Sentinel";
    case 8:
        return "Velthrax";
    case 9:
        return "Ulgarok";
    case 10:
        return "Molgrith";
    default:
        return "Vermin";
    }
}

static const std::string generateRoar()
{
    switch (Random::generateRandomNumber(1, 10))
    {
    case 1:
        return "Grrraahhh-rrrahhh!";
    case 2:
        return "RROOOAAARRR!";
    case 3:
        return "YII-HAH-RAAAAHH!";
    case 4:
        return "GRAAAH-HAHHH!";
    case 5:
        return "RAAAHHRRRRGHHH!";
    case 6:
        return "SHHHRRR-RAAAA!";
    case 7:
        return "GRRROOAAAARRRR!";
    case 8:
        return "RRAAAAGGGHHHHH!";
    case 9:
        return "SSS-KHHRRAAAAA!";
    case 10:
        return "WOOOOOOAAARRRRGHHH!";
    default:
        return "S-SSSSSSSS-RAAAH!";
    }
}

static const Monster generateRandomMonster()
{
    return { generateRace(), generateRandomName(), generateRoar(), Random::generateRandomNumber(1, 100) };
}

int main()
{
    Monster test{ generateRandomMonster() };
    displayMonster(test);

    return 0;
}
```

```cpp
#include <iostream>
#include <string>
#include <random>

// Simple Random utility namespace
namespace Random
{
    std::random_device rd;  // Non-deterministic random device
    std::mt19937 mt{ rd() }; // Mersenne Twister engine seeded with rd()

    // Generate random int between min and max (inclusive)
    int get(int min, int max)
    {
        std::uniform_int_distribution<> dist(min, max);
        return dist(mt);
    }
}

// Monster class
class Monster
{
public:
    // Nested unscoped enum Type for monster types
    enum Type
    {
        dragon,
        goblin,
        ogre,
        orc,
        skeleton,
        troll,
        vampire,
        zombie,
        maxMonsterTypes
    };

private:
    Type type;
    std::string name;
    std::string roar;
    int hitPoints;

public:
    // Constructor
    Monster(Type type, std::string name, std::string roar, int hitPoints)
        : type(type), name(std::move(name)), roar(std::move(roar)),
hitPoints(hitPoints) {}

    // Return monster type as string
    std::string getTypeString() const
    {
        switch (type)
        {
        case dragon:   return "dragon";
        case goblin:   return "goblin";
        case ogre:     return "ogre";
        case orc:      return "orc";
        case skeleton: return "skeleton";
        case troll:    return "troll";
        case vampire:  return "vampire";
        case zombie:   return "zombie";
        default:       return "unknown";
        }
    }

    // Print monster info
    void print() const
    {
        if (hitPoints > 0)
        {
```

```
            std::cout << name << " the " << getTypeString() << " has " << hitPoints << " hit points and says " << roar << ".\n";
        }
        else
        {
            std::cout << name << " the " << getTypeString() << " is dead.\n";
        }
    }
};

// MonsterGenerator namespace
namespace MonsterGenerator
{
    // Preset names
    std::string getName(int index)
    {
        static const std::string names[] = { "Blarg", "Mort", "Glim", "Nim", "Zed", "Rex" };
        if (index < 0 || index >= 6)
            return "Unknown";
        return names[index];
    }

    // Preset roars
    std::string getRoar(int index)
    {
        static const std::string roars[] = { "*ROAR*", "*growl*", "*hiss*", "*rattle*", "*grunt*", "*snarl*" };
        if (index < 0 || index >= 6)
            return "*silence*";
        return roars[index];
    }

    // Generate a random monster
    Monster generate()
    {
        Monster::Type type = static_cast<Monster::Type>(Random::get(0, Monster::maxMonsterTypes - 1));
        std::string name = getName(Random::get(0, 5));
        std::string roar = getRoar(Random::get(0, 5));
        int hitPoints = Random::get(1, 100);

        return Monster(type, name, roar, hitPoints);
    }
}

int main()
{
    Monster m = MonsterGenerator::generate();
    m.print();

    return 0;
}
```

0 Reply

Zeca

May 3, 2025 8:02 am PDT

took me one hour but there is it

```cpp
#include <iostream>
#include "Random.h"

class Monster
{
public:
    enum Type
    {
        dragon,
        goblin,
        ogre,
        orc,
        skeleton,
        troll,
        vampire,
        zombie,
        maxMonsterTypes
    };

private:
    Type m_type{};
    std::string m_name{};
    std::string m_roar{};
    int m_hp{};

public:
    Monster(Type type, std::string_view name, std::string_view roar, const int hp)
        : m_type{type}, m_name{name}, m_roar{roar}, m_hp{hp}
        {

        }

    std::string_view getTypeString()
    {
        switch(m_type)
        {
            case dragon:
                return "dragon";
            case goblin:
                return "goblin";
            case ogre:
                return "ogre";
            case orc:
                return "orc";
            case skeleton:
                return "skeleton";
            case troll:
                return "troll";
            case vampire:
                return "vampire";
            case zombie:
                return "zombie";
            default:
                return "???";
        }
    }

    void print(void)
    {
        if(m_hp == 0)
        {
            std::cout << m_name << " the " << getTypeString() << " is dead.\n";
        }
        else
            std::cout << m_name << " the " << getTypeString() << " has " << m_hp << " hit points and says " << m_roar << ".\n";
    }
};

namespace MonsterGenerator
```

```
{
    std::string_view getName(int a)
    {
        switch(a)
        {
            case 0:
                return "Blarg";
            case 1:
                return "Bones";
            case 2:
                return "Nibblez";
            case 3:
                return "Dragulon";
            case 4:
                return "Zarneth";
            case 5:
                return "Thirzagul";
            default:
                return "???";
        }
    }

    std::string_view getRoar(int a)
    {
        switch(a)
        {
            case 0:
                return "*ROAR*";
            case 1:
                return "*SKREEEEAAAARRGH*";
            case 2:
                return "*GRAAAAWRRRHH*";
            case 3:
                return "*KRAZAK'THUL*";
            case 4:
                return "*VROOOORNNN*";
            case 5:
                return "*ZHRRRAAAKKH*";
            default:
                return "???";
        }
    }

    Monster generate()
    {
        return {static_cast<Monster::Type>(Random::get(0,7)), getName(Random::get(0,5)), getRoar(Random::get(0,5)), Random::get(1, 100)};
    }
}

int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

0 Reply

Aayush

May 1, 2025 11:35 am PDT

Phew its finally working

```cpp
#include <iostream>
#include <string>
#include "Random.h"

class Monster
{
public:
    enum Type
    {
        dragon,
        goblin,
        ogre,
        orc,
        skeleton,
        troll,
        vampire,
        zombie,
        maxMonsterTypes,
    };

private:
    Type m_type{};
    std::string m_name{"???"};
    std::string m_roar{"???"};
    int m_hitPoints{};

public:
    Monster(Type type, std::string_view name, std::string_view roar, int hitPoints)
        : m_type{type}, m_name{name}, m_roar{roar}, m_hitPoints{hitPoints}
    {
    }

    constexpr std::string GetTypeString() const
    {
        switch (m_type)
        {
            case dragon: return "dragon";
            case goblin: return "goblin";
            case ogre: return "ogre";
            case orc: return "orc";
            case skeleton: return "skeleton";
            case troll: return "troll";
            case vampire: return "vampire";
            case zombie: return "zombie";
            case maxMonsterTypes: return "8";
        }

        return "???";
    }

    void print()
    {
        if (m_hitPoints <= 0)
        {
            std::cout << m_name << " the " << GetTypeString() << " is " << "dead " <<
"\n";
        }
        else
        {
            std::cout << m_name << " the " << GetTypeString() << " has " <<
m_hitPoints << " hit points and says " << m_roar << "\n";
        }
    }
};

namespace MonsterGenerator
{

    std::string_view getName(int name)
    {
```

```
        switch(name)
        {
            case 0: return "Blarg";
            case 1: return "wushang";
            case 2: return "lonewolf";
            case 3: return "gourav";
            case 4: return "aayush";
            case 5: return "adi";
            default: return "???";
        }
    }

    std::string_view getRoar(int roar)
    {
        switch(roar)
        {
            case 0: return "*ROAR*";
            case 1: return "nihao";
            case 2: return "aauuuu";
            case 3: return "shota";
            case 4: return "hentai";
            case 5: return "racist";
            default: return "???";
        }
    }

    Monster generate()
    {
        return Monster{static_cast<Monster::Type>(Random::get(0, 7)),
getName(Random::get(0, 5)), getRoar(Random::get(0, 5)), Random::get(1, 100)};
    }
}

int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

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

```cpp
#include <iostream>
#include <string>

class Monster
{
public:
    enum Type
    {
        Dragon,
        Goblin,
        Ogre,
        Orc,
        Skeleton,
        Troll,
        Vampire,
        Zombie,
        MaxMonsterTypes,
    };

private:
    Type m_type{};
    std::string m_name{"???"};
    std::string m_roar{"???"};
    int m_hitPoints{};

public:
    Monster (Type type, std::string& name, std::string& roar, int hitPoints)
        : m_type{type}
        , m_name{name}
        , m_roar{roar}
        , m_hitPoints{hitPoints}
    {
    }
};

int main()
{
    Monster skeleton { Monster::Skeleton, "Bones", "*rattle*", 4 };

    return 0;
}
```

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

```cpp
#ifndef MONSTER_H
#define MONSTER_H

#include <string>

class Monster
{
public:
    enum Type
    {
        dragon,
        goblin,
        orc,
        troll,
        ogre,
        skeleton,
        vampire,
        zombie,
        maxMonsterTypes
    };

    Monster(Type type, std::string_view name, std::string roar, int hitPoint)
        : m_type{ type }, m_name{ name }, m_roar{ roar }, m_hitPoint{ hitPoint }
    {
    }

    std::string getTypeString(Type type) const;

    void print() const;
private:
    Type m_type{};
    std::string m_name{"???"};
    std::string m_roar{"???"};
    int m_hitPoint{};
};


#endif MONSTER_H
```

```cpp
#include "Monster.h"
#include <iostream>


std::string Monster::getTypeString(Type type) const
{
    switch (type)
    {
        case dragon:
        return "dragon";
        break;
        case goblin:
        return "goblin";
        break;
        case orc:
        return "orc";
        break;
        case troll:
        return "troll";
        break;
        case ogre:
        return "ogre";
        break;
        case skeleton:
        return "skeleton";
        break;
        case vampire:
        return "vampire";
        break;
        case zombie:
        return "zombie";
        break;
        default:
            return "???";
            break;
    }
}

void Monster::print() const
{
    if (m_hitPoint > 0)
    {
        std::cout << m_name << " the "
            << getTypeString(m_type) << " has "
            << m_hitPoint << " hit points and says "
            << m_roar << ".\n";
    }
    else
    {
        std::cout << m_name << " the "
            << getTypeString(m_type) << " is dead.\n";
    }

}
```

main.cpp

```cpp
#include <string_view>
#include <iostream>
#include "Monster.h"
#include "Random.h"

namespace MonsterGenerator
{
    std::string_view getName(int index)
    {
        switch (index)
        {
        case 0:
            return "Blarg";
            break;
        case 1:
            return "Moog";
            break;
        case 2:
            return "Pksh";
            break;
        case 3:
            return "Tyrn";
            break;
        case 4:
            return "Mort";
            break;
        case 5:
            return "Hans";
            break;
        default:
            return "???";
            break;
        }
    }

    std::string getRoar(int index)
    {
        switch (index)
        {
        case 0:
            return "*RORA*";
            break;
        case 1:
            return "*peep*";
            break;
        case 2:
            return "*squeal*";
            break;
        case 3:
            return "*wine*";
            break;
        case 4:
            return "*growl*";
            break;
        case 5:
            return "*burp*";
            break;
        default:
            return "???";
            break;
        }
    }

    Monster generate()
    {
        int hitPoints = Random::get(1, 100);
        int name = Random::get(0, 5);
        int roar = Random::get(0, 5);
        int type = Random::get(0, 7);
        return Monster{ static_cast<Monster::Type>(type), getName(name),
```

```
getRoar(roar), hitPoints};
    }
}

int main()
{
    Monster m{ MonsterGenerator::generate() };
    m.print();

    return 0;
}
```

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