# S.7.x — Chapter 7 summary and quiz

BY ALEX ON JANUARY 3RD, 2020 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 7TH, 2020

## Quick review

std::string offers an easy and safe way to deal with text strings. String literals are always placed between double quotes.

Enumerated types let us define our own type where all of the possible values are enumerated. These are great for categorizing things.

Enum classes work like enums but offer more type safety and don't pollute the standard namespace quite as much.

And finally, structs offer us a way to group related variables into a single structure and access them using the member selection operator (.). Object-oriented programming builds heavily on top of these, so if you learn one thing from this chapter, make sure it's this one.

---

## Quiz time

Yay!

1) In designing a game, we decide we want to have monsters, because everyone likes fighting monsters. Declare a struct that represents your monster. The monster should have a type that can be one of the following: an ogre, a dragon, an orc, a giant spider, or a slime. Use an enum class for this.

Each individual monster should also have a name (use a std::string), as well as an amount of health that represents how much damage they can take before they die. Write a function named printMonster() that prints out all of the struct's members. Instantiate an ogre and a slime, initialize them using an initializer list, and pass them to printMonster().

Your program should produce the following output:

```
This Ogre is named Torg and has 145 health.
This Slime is named Blurp and has 23 health.
```

**Show Solution**



**5.1 -- Control flow introduction**


**Index**


**S.4.7 -- Structs**

C++ TUTORIAL | PRINT THIS POST

## 21 comments to S.7.x — Chapter 7 summary and quiz

giang
February 9, 2020 at 11:40 pm · Reply

Hi, I tried to write a program to solve the quiz above but my program works well when I just use 'enum' instead of 'enum class'. When I tried 'enum class'and '::' operator, the compile threw me errors that said:'MonsterType is not a class or namespace'. Can someone explain this situation to me, please? Thanksss

Lars
January 22, 2020 at 2:08 am · Reply

Here is my attempt on a solution.

monsters.h

```
#include <string>

#ifndef MONSTERS_H
#define MONSTERS_H

// This enum class contains the various types of monsters
// NB: To access a member, write MonsterType::DRAGON etc.
enum class MonsterType
{
    DRAGON,
    GIANT_SPIDER,
    GOBLIN,
    OGRE,
    ORC,
    SKELETON,
    SLIME,
    TROLL
};

// This struct defines a monster
struct Monster
{
```

```
    MonsterType type;   // The type of a monster is a member of enum class MonsterType
    std::string name;
    int health;         // health should be a positive integer
};

// namespace containing functions for printing description of a monster
namespace monsters
{
    std::string getStringMonsterType(Monster monster);  // returns type of a monster as a s
    void printMonster(Monster monster);                 // prints description of a monster
}

#endif // !MONSTERS_H
```

monsters.cpp

```
#include "monsters.h"
#include <iostream>

namespace monsters
{
    // Returns the type of a monster as a string
    // Used by the function printMonster
    std::string getStringMonsterType(Monster monster)
    {
        if (monster.type == MonsterType::DRAGON)
            return "dragon";
        else if (monster.type == MonsterType::GIANT_SPIDER)
            return "giant spider";
        else if (monster.type == MonsterType::GOBLIN)
            return "goblin";
        else if (monster.type == MonsterType::OGRE)
            return "ogre";
        else if (monster.type == MonsterType::ORC)
            return "orc";
        else if (monster.type == MonsterType::SKELETON)
            return "skeleton";
        else if (monster.type == MonsterType::SLIME)
            return "slime";
        else if (monster.type == MonsterType::TROLL)
            return "troll";

        return "Unknown";
    }

    // Prints a description of a monster
    void printMonster(Monster monster)
    {
        std::cout << "This " << getStringMonsterType(monster) << " is named " << monster.na
    }

}
```

main.cpp

```
#include "monsters.h"
#include <iostream>

// This program allows the user to create some monsters and print a description of them
int main()
{
    // Create monsters
```

```
    Monster torg{ MonsterType::OGRE, "Torg", 145 };
    Monster blurp{ MonsterType::SLIME, "Blurp", 23 };

    // Print description of monsters
    using monsters::printMonster;
    printMonster(torg);
    printMonster(blurp);

    return 0;
}
```

nascardriver
January 22, 2020 at 8:03 am · Reply

Looks good :)

Suyash
January 15, 2020 at 11:59 pm · Reply

Here is my solution to the quiz question... I have decided to split the code into 3 files: main.cpp, monsters.h and monsters.cpp files... The primary reason why I decided to modularize my code was to gain some extra practice by using header files to encapsulate different logical sections of a project.

The code works as expected, without any kind of compilation/linking problems... I would love to hear your opinions about it and you are more than welcome to bring into my observation anything that missed my eyes...

1. monsters.h

```
#ifndef MONSTERS_H
#define MONSTERS_H

#include <string>

enum class MonsterType
{
    OGRE,
    DRAGON,
    ORC,
    GIANT_SPIDER,
    SLIME
};

struct Monster
{
    MonsterType type;
    std::string name;
    int health;
};

namespace monsters
{
    std::string getTypeString(MonsterType type);
    void printMonster(Monster m);
}

#endif // !MONSTERS_H
```

2. monsters.cpp

```
#include <iostream>
```

```
#include "monsters.h"

namespace monsters
{
    std::string getTypeString(MonsterType type)
    {
        if (type == MonsterType::OGRE)
            return "Ogre";
        else if (type == MonsterType::DRAGON)
            return "Dragon";
        else if (type == MonsterType::ORC)
            return "Orc";
        else if (type == MonsterType::GIANT_SPIDER)
            return "Giant Spider";
        else
            return "Slime";
    }

    void printMonster(Monster m)
    {
        std::cout << "This " << monsters::getTypeString(m.type)
            << " is named " << m.name
            << " and has " << m.health << " health.\n";
    }
}
```

3. main.cpp

```
#include "monsters.h"

int main()
{
    Monster torg_the_ogre{ MonsterType::OGRE, "Torg", 145 };
    monsters::printMonster(torg_the_ogre);

    Monster blurp_the_slime{ MonsterType::SLIME, "Blurp", 23 };
    monsters::printMonster(blurp_the_slime);

    return 0;
}
```

nascardriver
January 16, 2020 at 2:33 am · Reply

Looks great, keep it up!

Lars
January 22, 2020 at 2:12 am · Reply

A bit more elegant than my attempt. I like it :)
One question:
In monsters.cpp line 22 is it best practice to specify namespace when calling getTypeString? More specifically, is it best practice to use
(a) monsters::getTypeString, or
(b) getTypeString
?

nascardriver
January 22, 2020 at 4:56 am · Reply

(b)
You're already in the namespace, there's no need to specify the namespace. Just like you don't use `::` whenever you access the global namespace and you don't say "on earth, on this continent, in this country, ..." every time someone asks you where something is.

fxr
January 15, 2020 at 1:57 pm · Reply

This is my attempt a the quiz I opted to have user input for monster creation any suggestions on how to improve it?

```
#include <iostream>
#include <string>
enum class Monster_Type // magic nums for monsters
{
    OGRE,
    DRAGON,
    ORC,
    GIANT_SPIDER,
    SLIME,

};

struct MonsterData // takes all the monster data
{
    std::string name{};
    int health{};
    Monster_Type type{};
};
void List() // prints list
{
    std::cout << "which of these types is your monster" << "\n"
    << "0. Ogre" << "\n" << "1. Dragon" << "\n" << "2. Orc"
    << "\n" << "3. Giant spider" << "\n" << "4. Slime" << "\n"
    << "answer with the number in front of the type " << "\n";
}
std::string NameDataIntake() // takes name
{
     std::string temp;
     using std::cin;
     using std::cout;
     cout << "what is you monsters name" << "\n";
     cin >> temp;
     return temp;
}
 int HealthData() // takes healt data
 {
     int temp;
     std::cout << " what is your monsters health" << "\n";
     std::cin >> temp;
     return temp;
 }
 Monster_Type MonsterDataCheck() // gets num for monster type
 {
     Monster_Type temp;
     List();
     int MonsterType;
```

```
    std::cin >> MonsterType;
    temp =static_cast<Monster_Type>(MonsterType) ; // converts to usable number
    return temp;
}
std::string MonsterIfs(Monster_Type type) // prints monster type
{
    using std::cout;
    if (type == Monster_Type::OGRE)
        return "OGRE";
    else if (type == Monster_Type::DRAGON)
        return "Dragon";
    else if (type == Monster_Type::ORC)
        return "ORC";
    else if (type == Monster_Type::GIANT_SPIDER)
        return "Giant Spider";
    else if (type == Monster_Type::SLIME)
        return "Slime";
    else
        return "ERROR";
}
int main()
{
    MonsterData Monster1 = { NameDataIntake(),HealthData(),MonsterDataCheck() }; // puts d
    std::cout << "This " << MonsterIfs(Monster1.type) <<
        " is named " << Monster1.name << " it has " << Monster1.health << " health. " << "
    return 0;
}
```

nascardriver
January 16, 2020 at 1:41 am · Reply

- Inconsistent formatting. Use your editor's auto-formatting feature.
- Initialize variables with brace initialization for better type safety.
- Use single quotation marks for characters ("\n" vs '\n')
- Line 21+: Magic numbers in strings, use your enum.
- Your function names aren't great. "listMonsters", "getMonsterName", "getMonsterHealth", "getMonsterType", "getMonsterTypeName" or similar are a lot more useful.

fxr
January 18, 2020 at 7:24 pm · Reply

thank you very much for the response will keep all of that in mind

fxr
January 19, 2020 at 10:54 am · Reply

Hello, it's me again nascardriver I was looking at my code improving the things you mentioned I have a question how could I use my Enum instead of magic numbers for the selection of a monster type.
Heres the part I'm talking about.

```
Monster_Type getMonsterType() // gets num for monster type
void List() // prints list
{
    std::cout << "which of these types is your monster" << "\n"
        << "0. Ogre" << "\n" << "1. Dragon" << "\n" << "2. Orc"
        << "\n" << "3. Giant spider" << "\n" << "4. Slime" << "\n"
```

```
        << "answer with the number in front of the type " << "\n";
}
{
    Monster_Type temp;
    List();
    int MonsterType;
    std::cin >> MonsterType;
    temp = static_cast<Monster_Type>(MonsterType); // converts to usable number
    return temp;
}
```

nascardriver
January 20, 2020 at 4:06 am · Reply

I thought you'd ask that, but I answered the same question the day before and couldn't find my reply, so I hoped you'd figure it out. You can cast your enumerators to `int` to make them printable, then just replace your magic strings with the `int`s.

```
std::cout << "which of these types is your monster\n"
          << static_cast<int>(Monster_Type::OGRE) << ". Ogre\n"
          << static_cast<int>(Monster_Type::Dragon) << ". Dragon\n"
          << /* ... */;
```

Now, when you move around your enumerators, the list is correct.

fxr
January 20, 2020 at 3:22 pm · Reply

Thank you so much nascardriver

kavin
January 12, 2020 at 1:32 am · Reply

I made one like this since its asked in the question to print out struct members. Its kind of big but, is there any way to make this simple ?

```
#include<iostream>

struct Monster
{
    std::string Aghora;
    std::string Hora;
    std::string Shiku;
    std::string Kushu;
    std::string Shikaku;
    double hpAghora;
    double hpHora;
    double hpShiku;
    double hpKushu;
    double hpShikaku;
};
enum class Montype
{
    OGRE,
    DRAGON,
    ORC,
```

```
    GIANT_SPYDER,
    SLIME,

};

void printMonster(Monster values, Montype ogre, Montype slime)
{
    //To print out all struct members
    std::cout << values.Aghora<<" "<<values.Hora<<" "<<values.Shiku<<" "<<values.Kushu<<" "
    std::cout << values.hpAghora << " " << values.hpHora << " " << values.hpShiku << " " <<
    std::cout << '\n';
    //To print out initialized ogre and slime
    if (ogre == Montype::OGRE)
        std::cout << "This Ogre is named " << values.Aghora << " and has " << values.hpAgho
    if (slime == Montype::SLIME)
        std::cout << "This Slime is named " << values.Shikaku << " and has " << values.hpSh
}

int main()
{
    Monster values{ "Aghora", "Hora", "Shiku", "Kushu", "Shikaku", 100, 95, 90, 85, 80 };

    //Instantiate an ogre and a slime
    Montype ogre = Montype::OGRE;
    Montype slime = Montype::SLIME;

    printMonster(values, ogre, slime);

    return 0;
}
```

nascardriver
January 12, 2020 at 4:38 am · Reply

The `struct` should describe the data that it holds, but not the values (eg. names of the monsters).

```
// We don't care what kind of monsters there will be, but they'll all
// have this structure.
struct Monster
{
  std::string name{};
  double hp{};
  MonsterType type{};
};

// ...

// Now we specify the values
Monster monsters[]{
  { "Aghora", 100, MonsterType::OGRE },
  { "Hora", 95, MonsterType::Slime },
  /* ... */
};
```

then pass only a `Monster` to `printMonster`, no other parameters.

Try rewriting your code that way. You seem to have misunderstood the purpose of structs and enums.

kavin
January 12, 2020 at 9:04 am · Reply

I am going nuts with this from morning and still can't figure out how to pass all values at once using just 1 argument like you said and print all the results in 1 function ! I understood the use of struct part you said but not the other thing .

```
enum class MonType
{
    OGRE,
    DRAGON,
    ORC,
    GIANT_SPYDER,
    SLIME
};
struct Monster
{
    MonType type{};
    std::string name{};
    double hp{};
};
//...
//...
int main()
{
    Monster monsters{ {MonType::OGRE, "Aghora", 100}
    {MonType::DRAGON, "Hora", 95}
    {MonType::ORC, "Shiku", 90}
    {MonType::GIANT_SPYDER, "Kushu", 85}
    {MonType::SLIME, "Skikaku", 80}
    };   //Is this correct ?

    printMonster(monsters); //How do i use this single argument and call a single
                            // struct members and specific montser related outpu
                            // understood the example where u passed ogre and sl
    return 0;
}
```

Sorry for being a super noob and firing series of questions at you. I don't want to move to next chapter without understanding this correctly !

nascardriver
January 13, 2020 at 1:40 am · Reply

> Is this correct ?
If you want to create multiple monsters, you need an array. You're declaring `monsters` as a single `Monster`, but trying to initialize it with multiple `Monster`s. That doesn't work.

```
Monster aghora{ MonsterType::OGRE, "Aghora", 100 }; // single monster

Monster monsters[]{
  { MonsterType::OGRE, "Aghora", 100  },
  { MonsterType::Slime, "Hora", 95 }
}; // 2 monsters

std::cout << aghora.name << " has " << aghora.hp << " hp\n";

std::cout << monsters[0].name << " has " << monsters[0].hp << " hp\n";
```

```
11  std::cout << monsters[1].name << " has " << monsters[1].hp << " hp\n";
```

Inside of `printMonster`, you can access the monster the same way.

```
1  void printMonster(Monster monster)
2  {
3    std::cout << monster.name << " has " << monster.hp << " hp\n";
4  }
```

kavin
January 14, 2020 at 12:46 am · Reply

Thank you so much for the replies @nascardriver Now i understand :) Its so helpful for me in learning the concepts completely.

hausevult
January 9, 2020 at 11:59 am · Reply

Woah, there are so few comments here! Well I like the way you use structs, really illuminating.

Kyle
January 6, 2020 at 5:03 am · Reply

There is no link to the next lesson here.

nascardriver
January 7, 2020 at 4:29 am · Reply

Link added, thanks!