5.x — Chapter 5 summary and quiz

Chapter Review

A **constant** is a value that may not be changed during the program's execution. C++ supports two types of constants: named constants, and literals.

A **named constant** is a constant value that is associated with an identifier. A **Literal constant** is a constant value not associated with an identifier.

A variable whose value can not be changed is called a **constant variable**. The **const** keyword can be used to make a variable constant. Constant variables must be initialized. Avoid using `const` when passing by value or returning by value.

A **type qualifier** is a keyword that is applied to a type that modifies how that type behaves. As of C++23, C++ only supports `const` and `volatile` as type qualifiers.

A **constant expression** is an expression that can be evaluated at compile-time. An expression that is not a constant expression is sometimes called a **runtime expression**.

A **compile-time constant** is a constant whose value is known at compile-time. A **runtime constant** is a constant whose initialization value isn't known until runtime.

A **constexpr** variable must be a compile-time constant, and initialized with a constant expression. Function parameters cannot be constexpr.

Literals are values inserted directly into the code. Literals have types, and literal suffixes can be used to change the type of a literal from the default type.

A **magic number** is a literal (usually a number) that either has an unclear meaning or may need to be changed later. Don't use magic numbers in your code. Instead, use symbolic constants.

In everyday life, we count using **decimal** numbers, which have 10 digits. Computers use **binary**, which only has 2 digits. C++ also supports **octal** (base 8) and **hexadecimal** (base 16). These are all examples of **numeral systems**, which are collections of symbols (digits) used to represent numbers.

The **conditional operator** (`?:`) (also sometimes called the **arithmetic if** operator) is a ternary operator (an operator that takes 3 operands). Given a conditional operation of the form `c ? x : y`, if conditional `c` evaluates to `true` then `x` will be evaluated, otherwise `y` will be evaluated. The conditional operator typically needs to be parenthesized as follows:

- Parenthesize the entire conditional operator when used in a compound expression (an expression with other operators).
- For readability, parenthesize the condition if it contains any operators (other than the function call operator).

Inline expansion is a process where a function call is replaced by the code from the called function's definition. A function that is declared using the `inline` keyword is called an **inline function**.

Inline functions and variables have two primary requirements:

- The compiler needs to be able to see the full definition of an inline function or variable in each translation unit where the function is used (a forward declaration will not suffice on its own). The definition can occur after the point of use if a forward declaration is also provided.
- Every definition for an inline function or variable must be identical, otherwise undefined behavior will result.

In modern C++, the term inline has evolved to mean "multiple definitions are allowed". Thus, an inline function is one that is allowed to be defined in multiple files. C++17 introduced **inline variables**, which are variables that are allowed to be defined in multiple files.

Inline functions and variables are particularly useful for **header-only libraries**, which are one or more header files that implement some capability (no .cpp files are included).

A **constexpr function** is a function whose return value may be computed at compile-time. To make a function a constexpr function, we simply use the `constexpr` keyword in front of the return type. Constexpr functions are only guaranteed to be evaluated at compile-time when used in a context that requires a constant expression. Otherwise they may be evaluated at compile-time (if eligible) or runtime.

A **consteval function** is a function that must evaluate at compile-time.

Constexpr functions and consteval functions are implicitly inline.

A **string** is a collection of sequential characters that is used to represent text (such as names, words, and sentences). String literals are always placed between double quotes. String literals in C++ are C-style strings, which have a strange type that is hard to work with.

std::string offers an easy and safe way to deal with text strings. std::string lives in the <string> header. `std::string` is expensive to initialize (or assign to) and copy.

std::string_view provides read-only access to an existing string (a C-style string literal, a std::string, or a char array) without making a copy. A `std::string_view` that is viewing a string that has been destroyed is sometimes called a **dangling** view. When a `std::string` is

modified, all views into that `std::string` are **invalidated**, meaning those views are now invalid. Using an invalidated view (other than to revalidate it) will produce undefined behavior.

Because C-style string literals exist for the entire program, it is okay to set a `std::string_view` to a C-style string literal, and even return such a `std::string_view` from a function.

A **substring** is a contiguous sequence of characters within an existing string.

Quiz time

Question #1

Why are named constants often a better choice than literal constants?

Why are const/constexpr variables usually a better choice than #defined symbolic constants?

Show Solution

Question #2

Find 3 issues in the following code:

```
#include <cstdint> // for std::uint8_t
#include <iostream>

int main()
{
  std::cout << "How old are you?\n";

  std::uint8_t age{};
  std::cin >> age;

  std::cout << "Allowed to drive a car in Texas: ";

  if (age >= 16)
      std::cout << "Yes";
  else
      std::cout << "No";

  std::cout << '.\n';

  return 0;
}
```

Sample desired output:

```
How old are you?
6
Allowed to drive a car in Texas: No

How old are you?
19
Allowed to drive a car in Texas: Yes
```

Show Solution

Question #3

Add `const` and/or `constexpr` to the following program:

```
// gets tower height from user and returns it
double getTowerHeight()
{
	std::cout << "Enter the height of the tower in meters: ";
	double towerHeight{};
	std::cin >> towerHeight;
	return towerHeight;
}

// Returns ball height from ground after "seconds" seconds
double calculateBallHeight(double towerHeight, int seconds)
{
	double gravity{ 9.8 };

	// Using formula: [ s = u * t + (a * t^2) / 2 ], here u(initial velocity) = 0
	double distanceFallen{ (gravity * (seconds * seconds)) / 2.0 };
	double currentHeight{ towerHeight - distanceFallen };

	return currentHeight;
}

// Prints ball height above ground
void printBallHeight(double ballHeight, int seconds)
{
	if (ballHeight > 0.0)
		std::cout << "At " << seconds << " seconds, the ball is at height: "
<< ballHeight << " meters\n";
	else
		std::cout << "At " << seconds << " seconds, the ball is on the
ground.\n";
}

// Calculates the current ball height and then prints it
// This is a helper function to make it easier to do this
void printCalculatedBallHeight(double towerHeight, int seconds)
{
	double ballHeight{ calculateBallHeight(towerHeight, seconds) };
	printBallHeight(ballHeight, seconds);
}

int main()
{
	double towerHeight{ getTowerHeight() };

	printCalculatedBallHeight(towerHeight, 0);
	printCalculatedBallHeight(towerHeight, 1);
	printCalculatedBallHeight(towerHeight, 2);
	printCalculatedBallHeight(towerHeight, 3);
	printCalculatedBallHeight(towerHeight, 4);
	printCalculatedBallHeight(towerHeight, 5);
```

```
        return 0;
}
```

Show Solution

Question #4

What are the primary differences between `std::string` and `std::string_view`?

What can go wrong when using a `std::string_view`?

Show Solution

Question #5

Write a program that asks for the name and age of two people, then prints which person is older.

Here is the sample output from one run of the program:

```
Enter the name of person #1: John Bacon
Enter the age of John Bacon: 37
Enter the name of person #2: David Jenkins
Enter the age of David Jenkins: 44
David Jenkins (age 44) is older than John Bacon (age 37).
```

Show Hint

Show Solution

Question #6

Complete the following program:

```
#include <iostream>

// Write the function getQuantityPhrase() here

// Write the function getApplesPluralized() here

int main()
{
    constexpr int maryApples { 3 };
    std::cout << "Mary has " << getQuantityPhrase(maryApples) << ' ' <<
getApplesPluralized(maryApples) << ".\n";

    std::cout << "How many apples do you have? ";
    int numApples{};
    std::cin >> numApples;

    std::cout << "You have " << getQuantityPhrase(numApples) << ' ' <<
getApplesPluralized(numApples) << ".\n";

    return 0;
}
```

Sample output:

```
Mary has a few apples.
How many apples do you have? 1
You have a single apple.
```

`getQuantityPhrase()` should take a single int parameter representing the quantity of something and return the following descriptor:

- < 0 = “negative”
- 0 = “no”
- 1 = “a single”
- 2 = “a couple of”
- 3 = “a few”
- > 3 = “many”

`getApplesPlural()` should take a single int parameter parameter representing the quantity of apples and return the following:

- 1 = “apple”
- otherwise = “apples”

This function should use the conditional operator.

Both functions should make proper use of `constexpr`.

Show Hint

Show Solution