

Slides

Development > Programming Languages > C++

The C++ 20 Masterclass : From Fundamentals to Advanced

Learn and Master Modern C++ From Beginning to Advanced in Plain English : C++11, C++14, C++17, C++20 and More!

4.7 ★★★★★

Created by [Daniel Gakwaya](#)

Section : Literals and constants

Slide intentionally left empty

Literals and Constants : Introduction

Literal

Data that is directly represented in code without going through some other variable stored in memory

Literal

They are literally stored in the program executable file, hence the name literal!

const

A read only variable. Can't assign data to it

constexpr

A constant that has the POTENTIAL to be evaluated at compile time

constinit

A variable that should be initialized with a constant or literal at compile time

Slide intentionally left empty

Literals

Literal

Data that is directly represented in code without going through some other variable stored in memory

Literal

They are literally stored in the program executable file, hence the name literal!

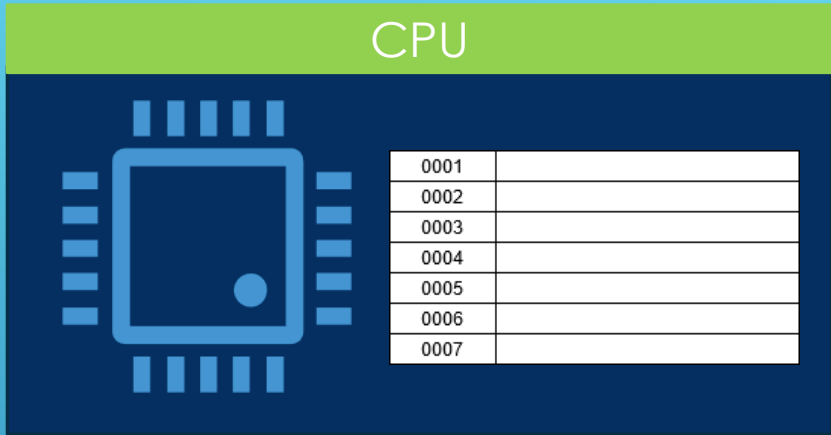
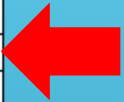
```
int int_var {55} ;
```



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1
0	1	The C++ 20 Masterclass: From Fundamentals to Advanced © Daniel Gakwaya																							1	1	1	1		
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1

Program
area

0001	a= 10 (int)
0002	b = 5 (int)
0003	c (int)
0004	print("Statement1")
0005	print("Statement2")
0006	c = f_add(a,b)
0007	print("Statement3")
0008	print("Statement4")
0009	end
0010	
...	
...	
0020	10
...	5
...	
0030	
...	
	Param1 + param2
...	
...	



Hard Drive



```
a = 10 (int)
b = 5 (int)
c (int)
print("Statement1")
print("Statement2")
c = f_add(a,b)
print("Statement3")
print("Statement4")
end
```

```
//Literal types : u and l combinations for unsigned and long.
unsigned char unsigned_char {53u}; // 555U would fail because of narrowing

//2 Bytes
short short_var {-32768} ; // No special literal type for short)
short int short_int {455} ; // No special literal type for short
signed short signed_short {122}; // No special literal type for short
signed short int signed_short_int {-456}; // No special literal type for short
unsigned short int unsigned_short_int {5678U } ;

// 4 Bytes
const int int_var {55} ; //
signed signed_var {66}; //
signed int signed_int {77}; //
unsigned int unsigned_int {555U}; //
```

```
//4 or 8 Bytes
long long_var {88L}; // 4 OR 8 Bytes
long int long_int {33L};
signed long signed_long {44l};
signed long int signed_long_int {44l};
unsigned long int unsigned_long_int {555ul};

long long long_long {888ll}; // 8 Bytes
long long int long_long_int {999ll};
signed long long signed_long_long {444ll};
signed long long int signed_long_long_int{1234ll};
```



```
//Grouping Numbers : C++14 and onwards
unsigned int prize {1'500'00'0u};
std::cout << "The prize is : " << prize << std::endl;
```

```
//Possible narrowing errors are caught by the braced initializer method.  
//Assignment and functional don't catch that.  
unsigned char distance { 555u}; //Error  
unsigned int game_score {-20}; //Error
```

```
//With number systems - Hex : prefix with 0x
unsigned int hex_number{ 0x22BU}; // Dec 555
int hex_number2 {0x400}; // Dec 1024
std::cout << "The hex number is : " << hex_number << std::endl;
std::cout << "The hex number2 is : " << hex_number2 << std::endl;

//Representing colors with hex
int black_color {0xffffffff};
std::cout << "Black color is : " << black_color << std::endl;
```

```
//Octal literals : prefix with 0
int octal_number {0777u}; // 511 Dec
std::cout << "The octal number is : " << octal_number << std::endl;
//!!!BE CAREFUL NOT TO PREFIX YOUR INTEGERS WITH 0 IF YOU MEAN DEC
int error_octal {055}; // This is not 55 in memory , it is 45 dec
std::cout << "The erroneous octal number is : " << error_octal << std::endl;

//Binary literals
unsigned int binary_literal {0b11111111u}; // 255 dec
std::cout << "The binary literal is : " << binary_literal << std::endl;
```

```

1 // Other literals. This is just an example and we will learn
2 // more about strings as we progress in the course.
3 char char_literal {'c'};
4 int number_literal {15};
5 float fractional_literal {1.5f};
6 std::string string_literal {"Hit the road"};
7
8 std::cout << "The char literal is : " << char_literal << std::endl;
9 std::cout << "The number literal is : " << number_literal << std::endl;
10 std::cout << "The fractional literal is : " << fractional_literal << std::endl;
11 std::cout << "The string literal is : " << string_literal << std::endl;
12

```

Slide intentionally left empty

Constants

Constant

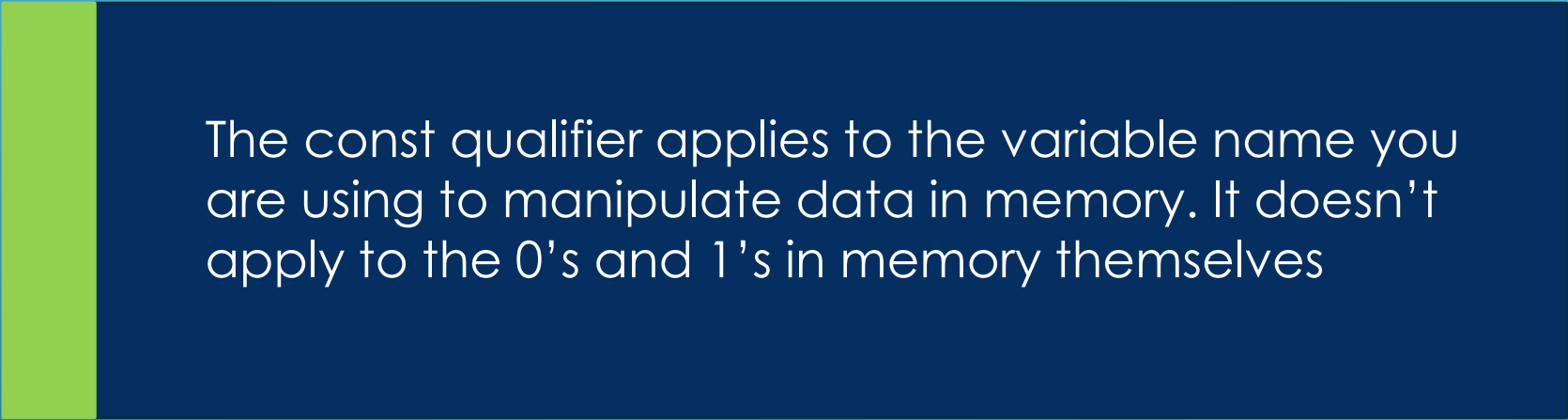
A variable you can initialize, but can't change afterwards


```
const unsigned int earth_radius_km { 6371 } ; // Radius in km
```

```
const unsigned int earth_radius_km { 6371 } ; // Radius in km
```



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1	1



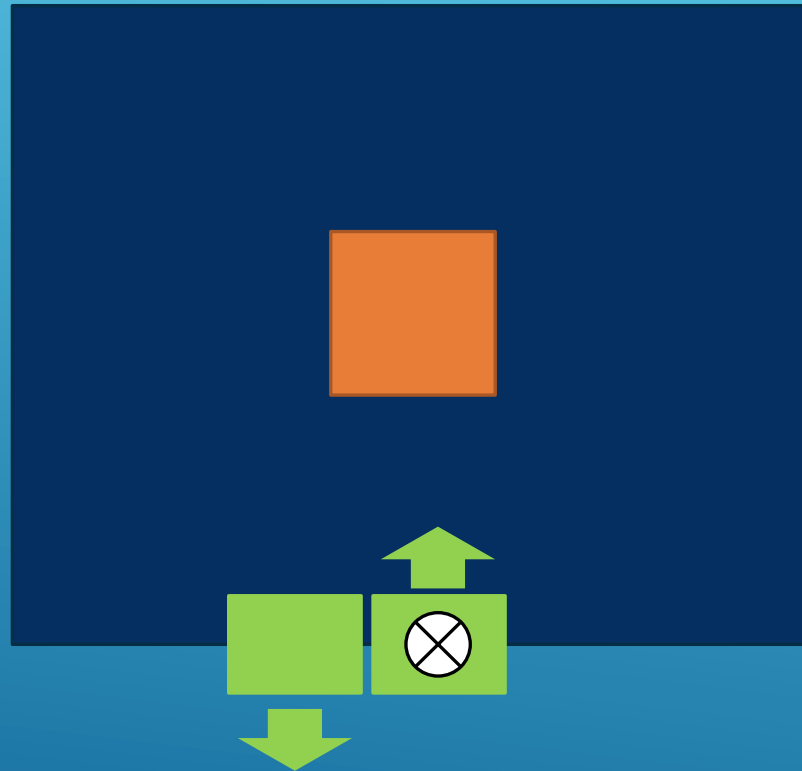
The const qualifier applies to the variable name you are using to manipulate data in memory. It doesn't apply to the 0's and 1's in memory themselves

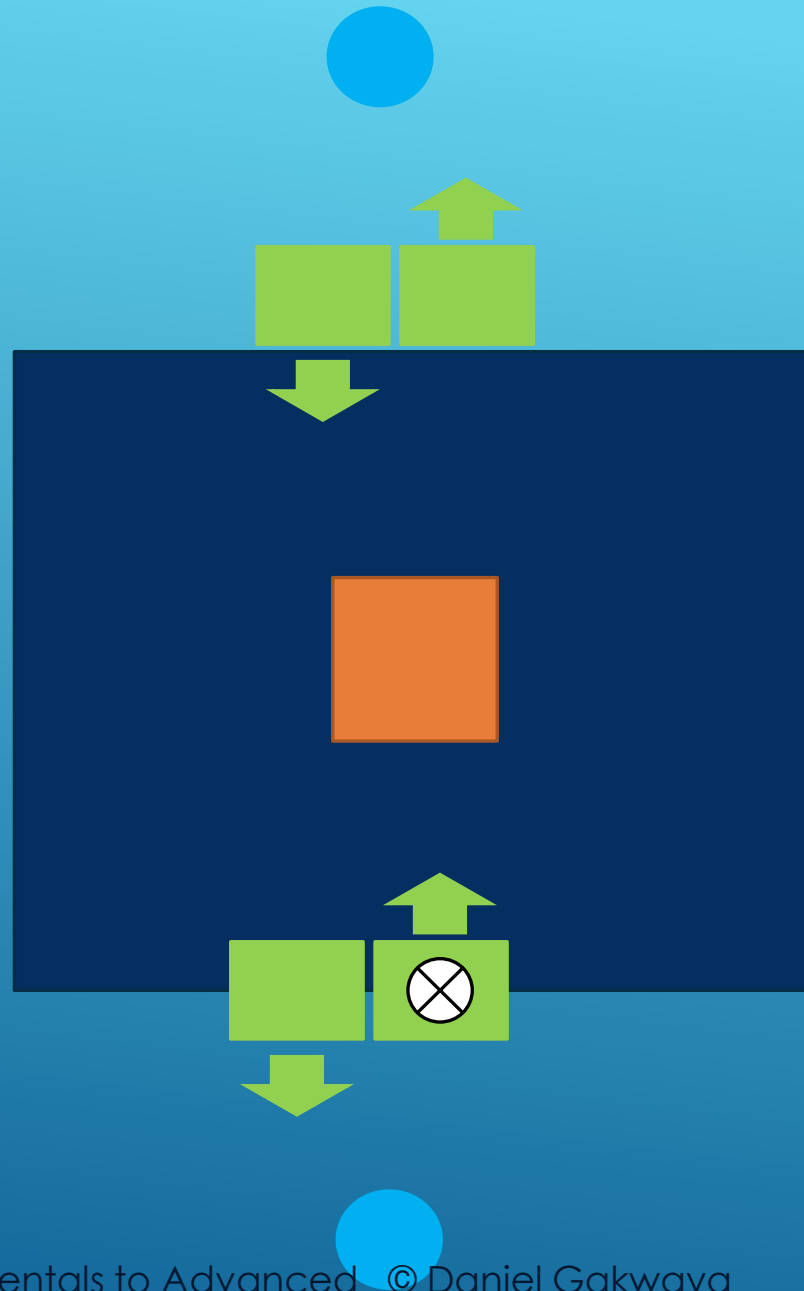
```
const unsigned int earth_radius_km { 6371 } ; // Radius in km
```

other_variable



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1





Why const ?

Guarantee that the variable will never change
throughout the lifetime of your program


```
//Declare and initialize  
const unsigned int earth_radius_km { 6371 } ; // Radius in  
const unsigned int earth_radius_km_1 ( 6371) ; // Radius in km  
const unsigned int earth_radius_km_2 = 6371 ; // Radius in km
```

```
// Can't declare un-initialized  
const unsigned int earth_radius_km ; // Compiler error
```

```
//Can't modify the data in a const variable  
const unsigned int earth_radius_km { 6371 } ;  
earth_radius_km = 7000; //Compiler error  
earth_radius++; // Compiler error
```

```
//Can be involved in other calculations
const unsigned int earth_radius_km { 6371 } ;
int expanded_radius = 3 * earth_radius_km;

std::cout << " Expanded earth radius : " << expanded_radius << std::endl;
```



int

const int

When to use const ?

- Using const where it makes sense in your code makes it self documenting in that when someone sees your variable declaration, they instantly know that it is a read only piece of data
- You also get the compiler protection when you try to modify the read only variable by mistake
- It is recommended to make your variables const where possible
- I personally declare most of my variables const and then take the const out when I need the variable to be modified.

Slide intentionally left empty

Constant Expressions

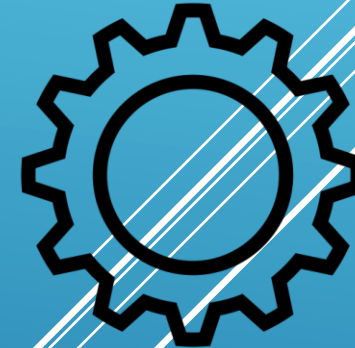
IDE

```
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}
```

Compile Time

Compiler



Run Time

Executable binary file

constexpr

Constant that may be evaluated at compile time or runtime

- If possible, move the potentially heavy computations at compile time
- The heavy computation is done once by the developer and users running the application can benefit from the results of the computation done at compile time

```
#include <iostream>

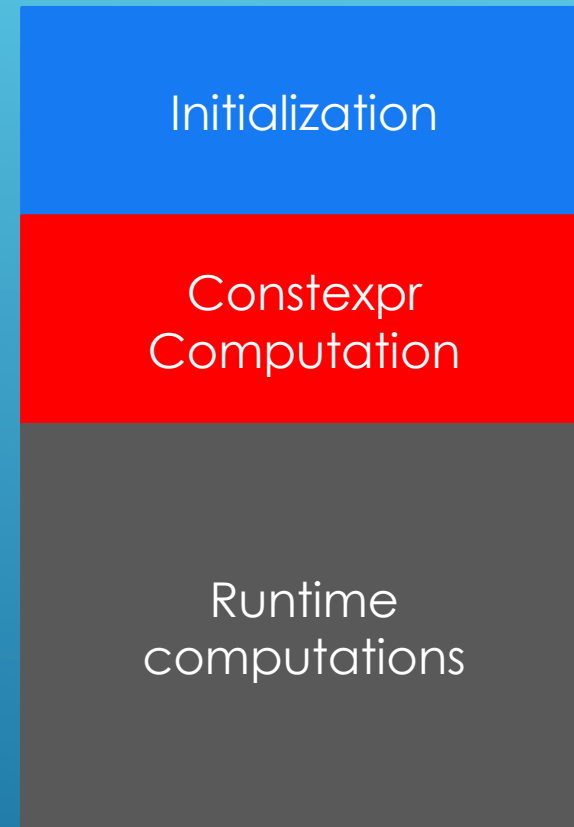
int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}
```

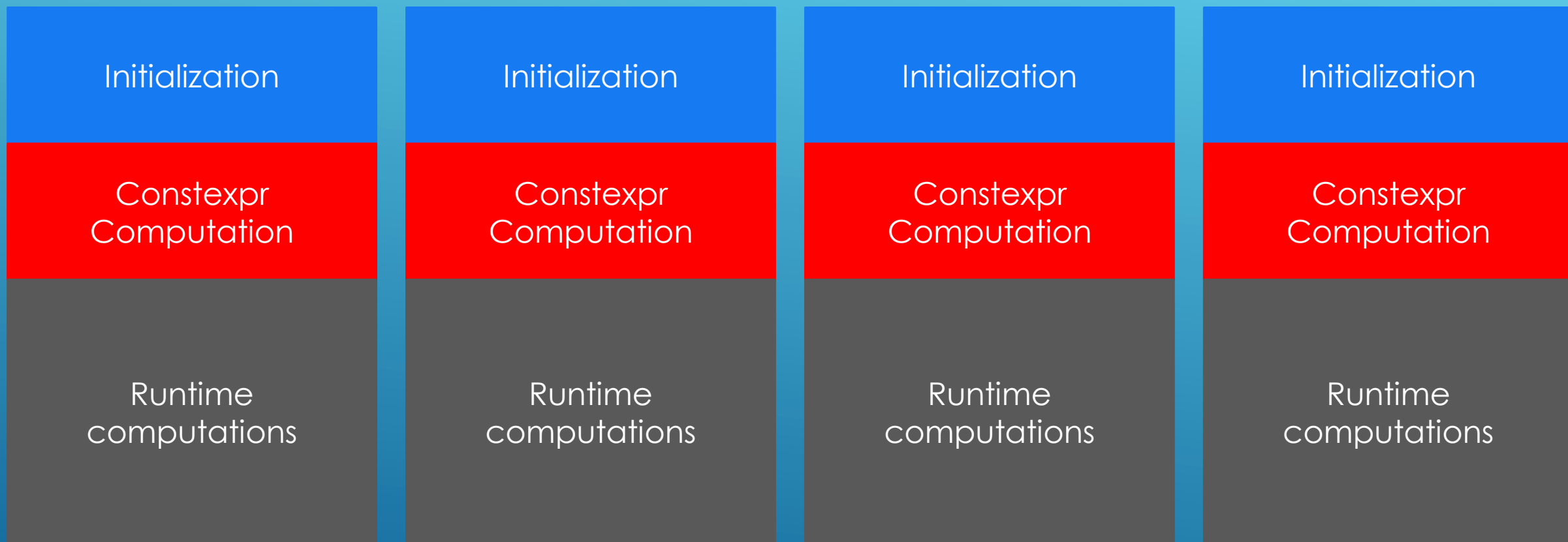


Compiler



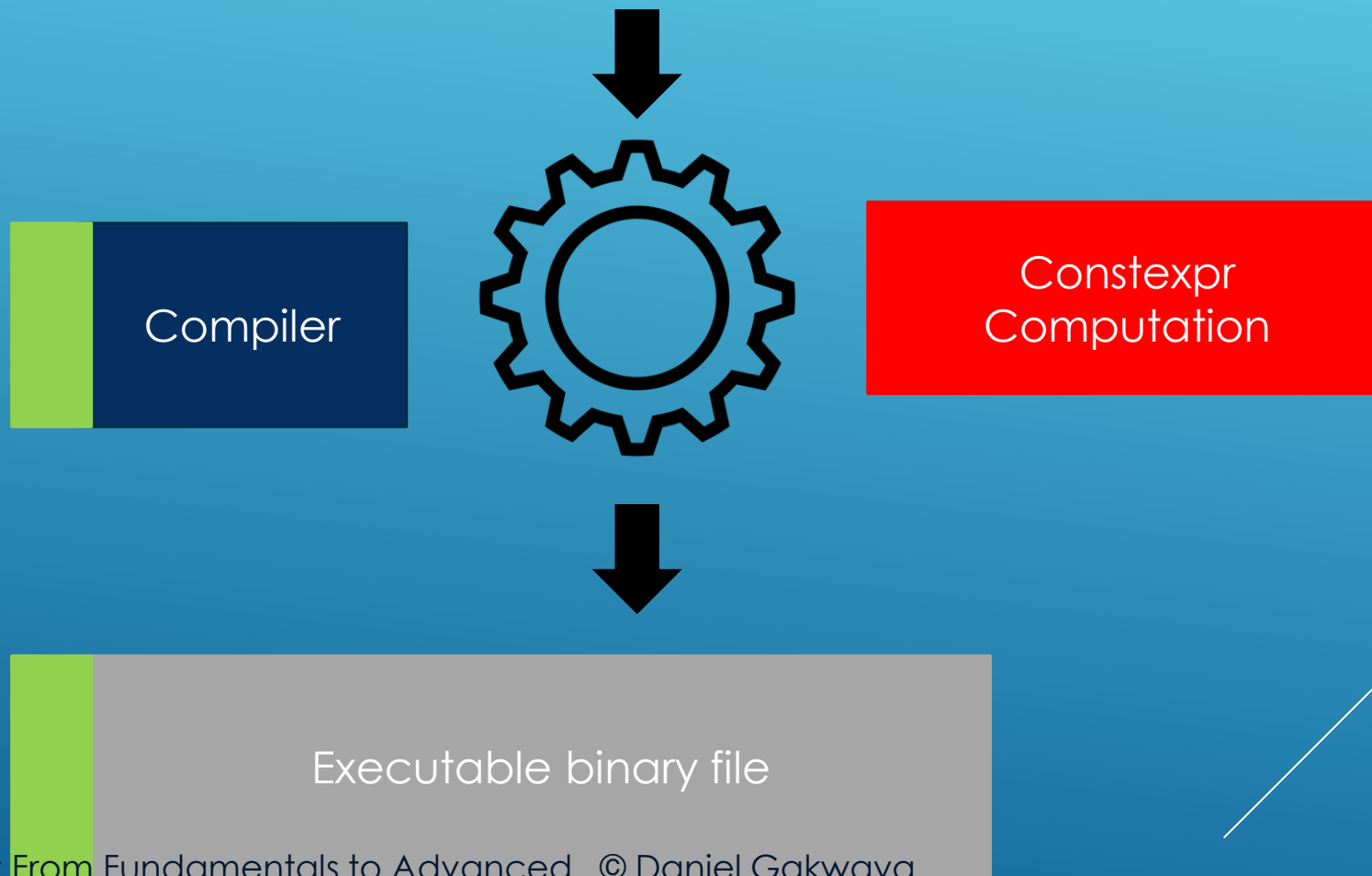
Executable binary file



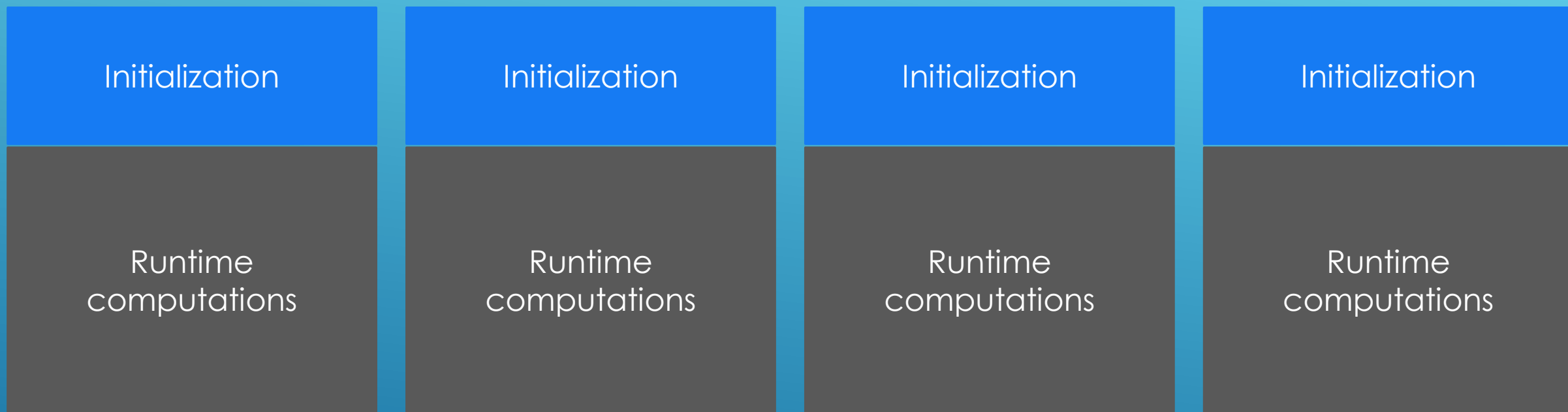


```
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}
```







Constant expressions have been introduced in C++11
Before that, everything was done at runtime
With every new iteration of the C++ standard, we see the range of things we can do at compile time increase.

Runtime computations

Compile time computations

Runtime computations

Compile time computations

Runtime computations

Compile time computations

Runtime computations

Compile time computations

Runtime computations

Compile time computations

Runtime computations

```
constexpr int eye_count {2};

constexpr double PI {3.14};

std::cout << "eye count : " << eye_count << std::endl;
std::cout << "PI : " << PI << std::endl;

int leg_count {2}; // Non constexpr
// leg_count is not known at compile time
constexpr int arm_count{leg_count}; // Error

constexpr int room_count{10};
constexpr int door_count{room_count}; // OK

const int table_count{5};
constexpr int chair_count{ table_count * 5}; // Works
```

Checks at compile time

```
static_assert(eye_count == 2);  
static_assert(SOME_LIB_MAJOR_VERSION == 123);
```



Constant expressions are also constants, so
you can't reassign values to them



Slide intentionally left empty



constinit

```
10. constexpr says that a variable should be initialized at compile
11. time. It's a new C++20 keyword

12. If you try to initialize with something that can't be evaluated at
13. compile time, you'll get a compiler error.
14. We say that the variable should be const initialized

15. constexpr can only be applied to variables with static or thread storage
16. duration. This, in part means variables outside the scope of the main
17. function. We'll understand more about this later in the course when
18. we have more powerful tools in our hands.

19. constexpr is in place in part to help in avoiding problems with the
20. order of initialization of global variables outside the main function

21. constexpr variables must be initialized with const or literals.

22. const and constexpr can be combined, but const and constexpr can't be
23. combined in an expression.

24. CAREFUL HERE : const init doesn't imply that the variable is const .
25. It just implies that the compiler enforces initialization
26. at compile time.
```

```

#include <iostream>

const int val1 {33};
constexpr int val2{34};
int val3 {35};

constinit int age = 88;
const constinit int age1 {val1}; // const and constinit can be combined
constinit int age2 {age1}; // Initializing with age would lead to a compiler error
//constexpr int age3 {age1}; // age is not const
//constinit int age3 {val3}; // Error : val3 is evaluated at run time
//constexpr int age3 {val3}; // can't const initialize age3

const constinit double weight {33.33};
//constexpr constinit double scale_factor{3.11}; // Can't combine constexpr and constinit

int main(int argc, char **argv)
{
    //constexpr int age4{41}; // Compiler error : not static or thread local storage
    /* ...
    return 0;
}

```


Slide intentionally left empty

Section Summary

Literals and constants

Literal

Data that is directly represented in code without going through some other variable stored in memory

Constant

A variable you can initialize, but can't change afterwards

`constexpr`

`static_assert()`

`constinit`

70

Slide intentionally left empty