- ✓ Classes & Objects
- ✓ Encapsulation & Abstraction
- ✓ Inheritance & Polymorphism
- ✓ RAII & Smart Pointers
- ✓ Templates & STL
- ✓ vector · map · algorithms
- ✓ Exceptions
- ✓ Data Structures · Modern C++

C++

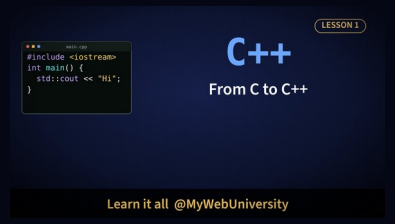
FULL COURSE

17 Lessons · Beginner to Advanced

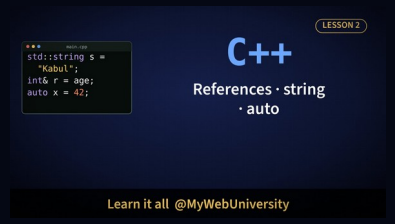
Free · English & Dari

Learn it all @MyWebUniversity

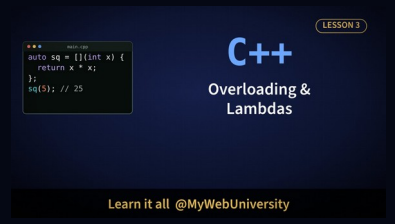
The 17 Lessons



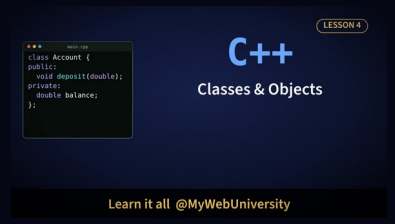
1. From C to C++



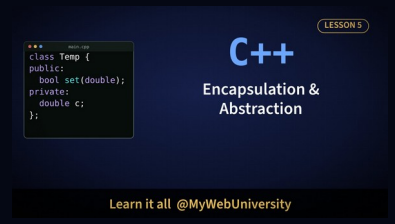
2. C++ Basics: Types, References, string and auto



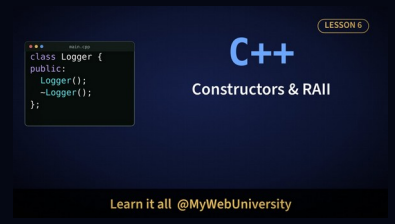
3. Functions, Overloading and Lambdas



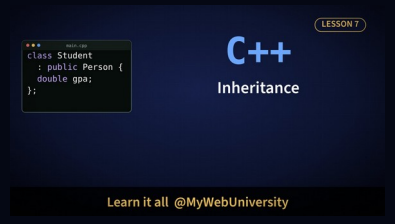
4. Classes and Objects



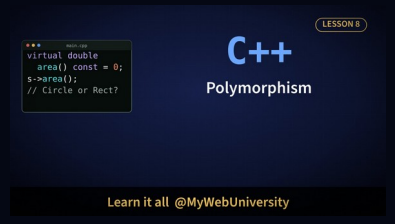
5. Encapsulation, Data Hiding and Abstraction



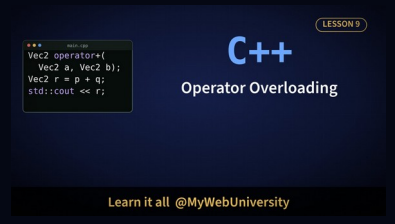
6. Constructors, Destructors and RAII



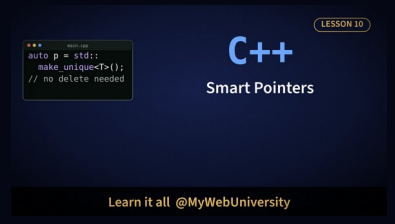
7. Inheritance



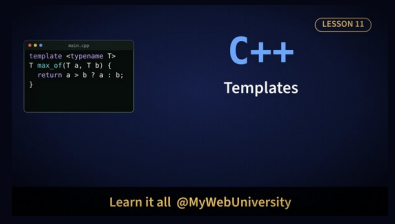
8. Polymorphism and Virtual Functions



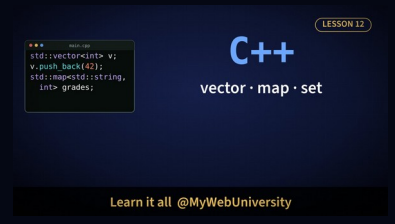
9. Operator Overloading



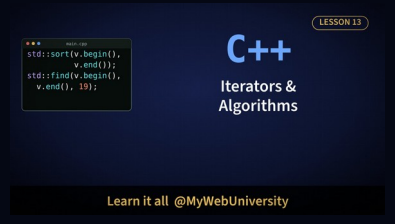
10. Memory and Smart Pointers



11. Templates and Generic Programming



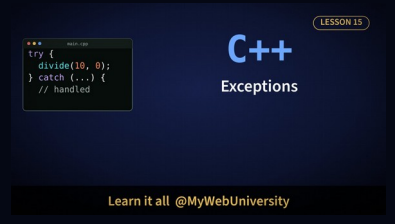
12. STL Containers: vector, map and More



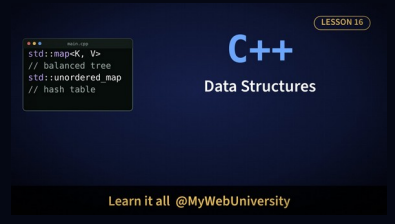
13. Iterators and Algorithms



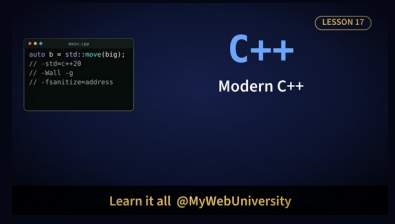
14. Strings and File Streams



15. Exceptions and Error Handling



16. Data Structures in C++



17. Modern C++, Projects and Next Steps



A FREE MYWEBUNIVERSITY COURSE

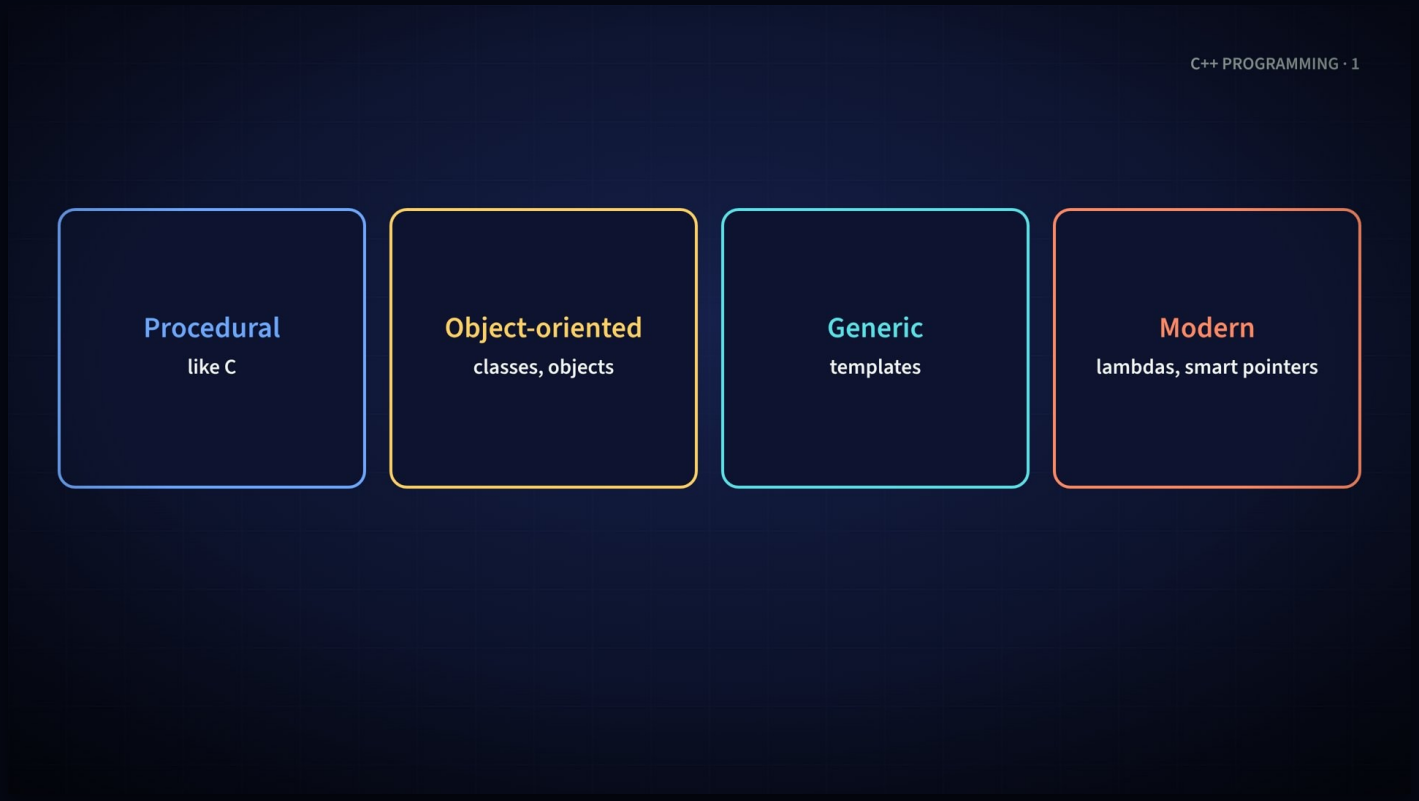
C++ PROGRAMMING

LESSON 1

From C to C++

What C++ adds, the standards, and your first program

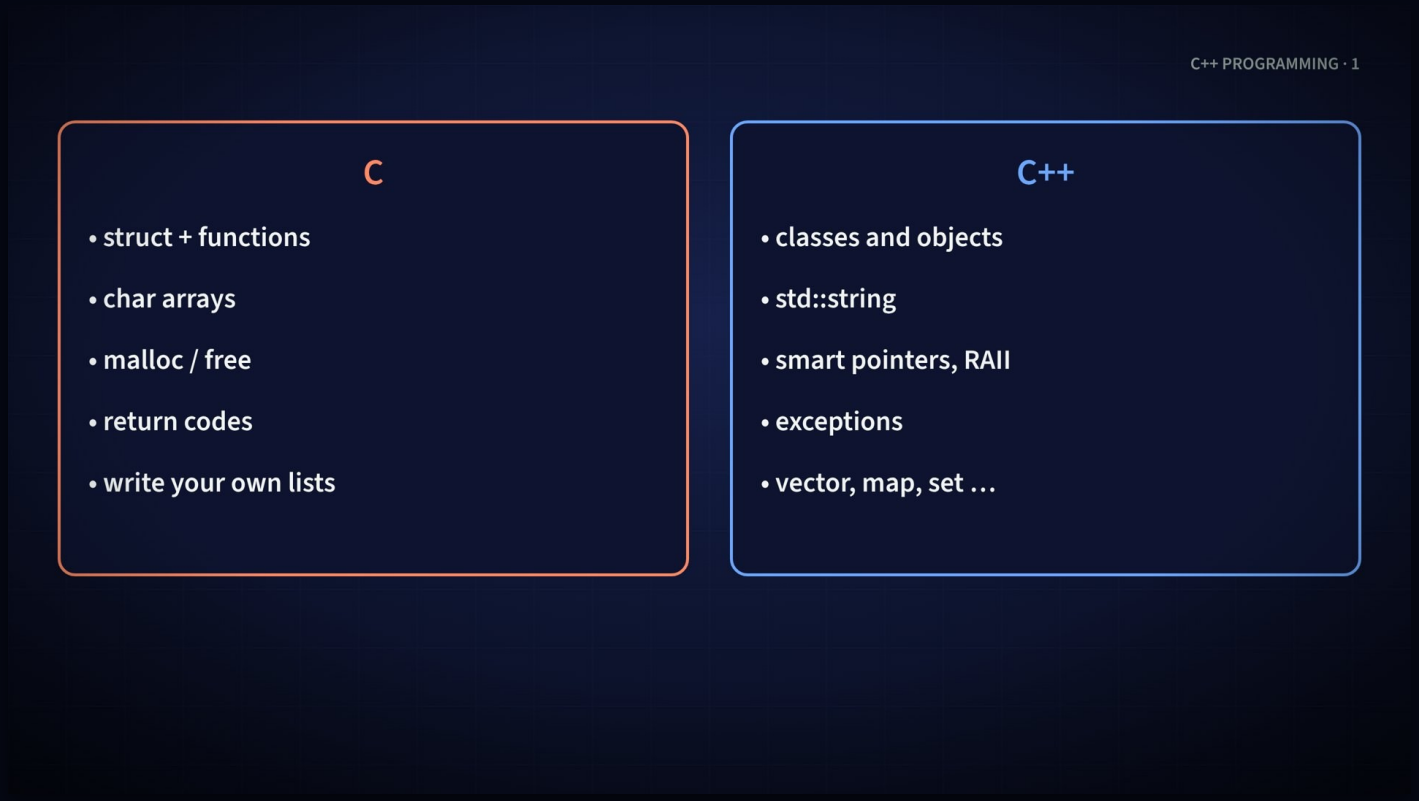
What Is C++?



- Procedural — like C
- Object-oriented — classes, objects
- Generic — templates

1.01

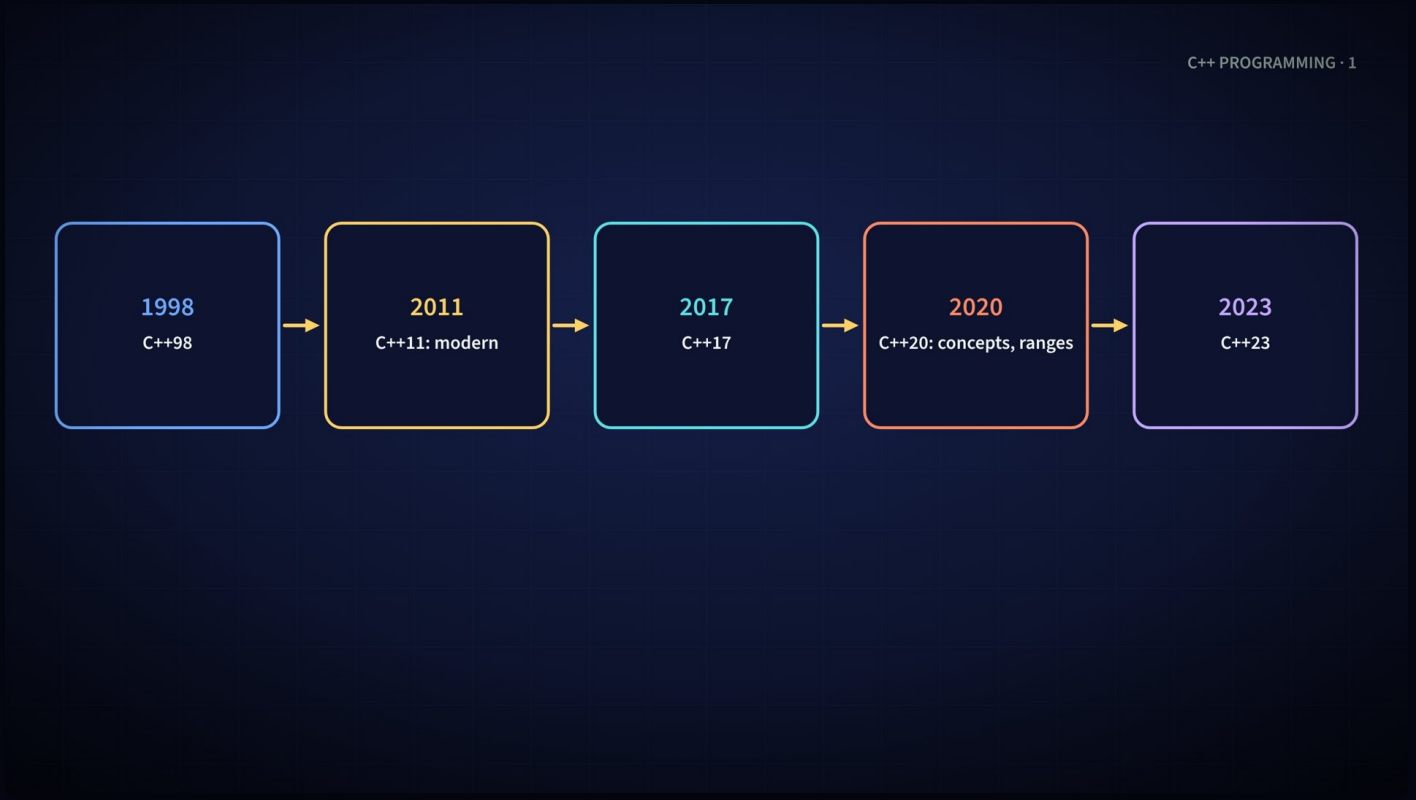
What C++ Adds to C



- C plus plus adds classes and objects, so data and behavior live together. It adds …
- Its standard library gives you ready-made containers like vector and map, a real string …
- Almost all C code is also valid C plus plus, so everything you learned in the C course …

1.02

C++ Standards



- 1998 — C++98
- 2011 — C++11: modern
- 2017 — C++17

1.03

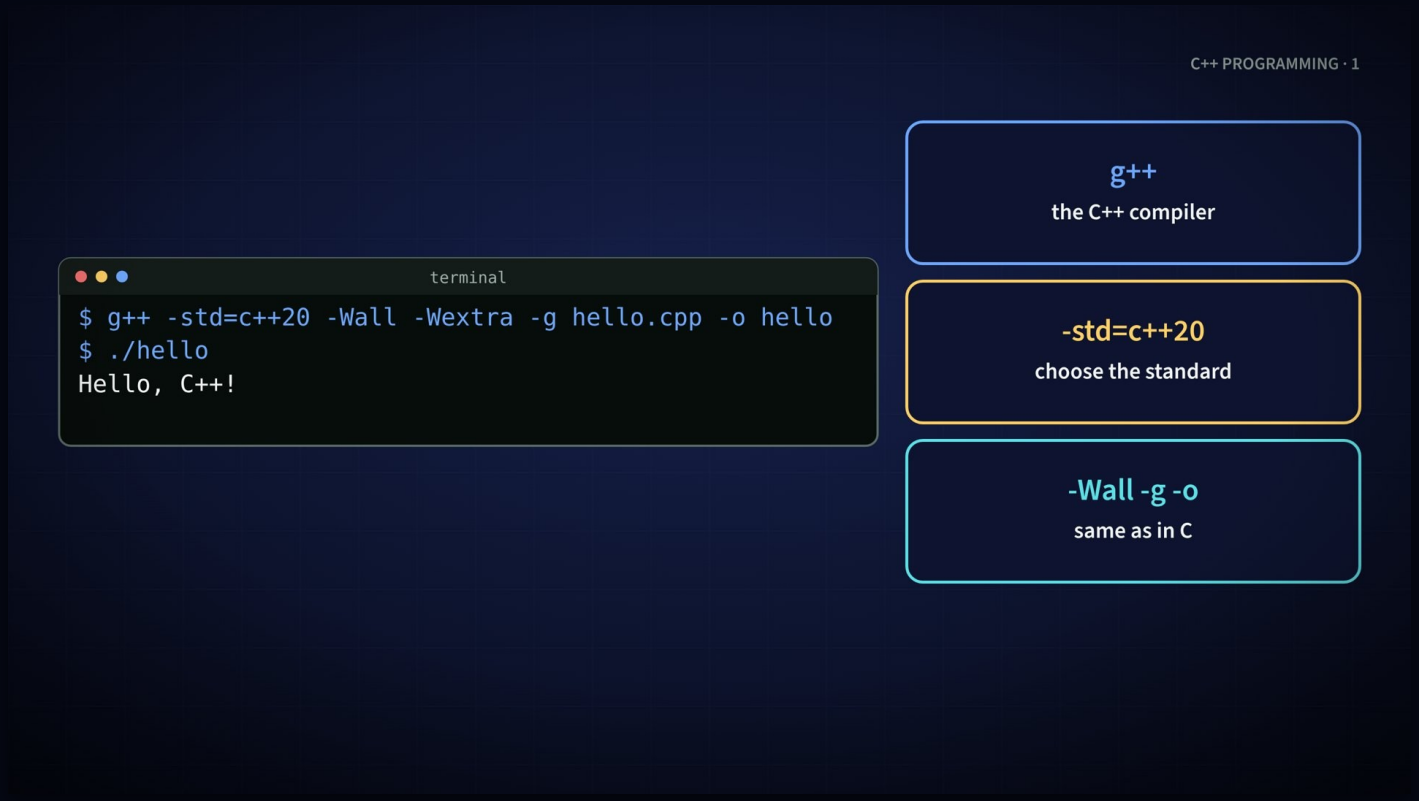
Hello, C++



- <iostream> — input and output
- std::cout << — print to the screen
- std:: — standard namespace

1.04

Compiling with g++



- g++ — the C++ compiler
- -std=c++20 — choose the standard
- -Wall -g -o — same as in C

1.05

Input with cin

input.cpp

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

int main() {
    std::string name;
    int age;
    std::cout << "Full name: ";
    std::getline(std::cin, name);
    std::cout << "Age: ";
    std::cin >> age;
    std::cout << "Hello, " << name << "! Next year: " << age + 1 << "\n";
}
```

- std colon colon cin reads input, using two greater-than signs. It knows the type of each …
- To read a whole line, including spaces, use getline with a std string.

1.06

Course Roadmap



- 1–3 — basics · functions
- 4–9 — classes · the 4 pillars
- 10–13 — memory · templates · STL

1.07



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 2

C++ Basics: Types, References, string and auto

The everyday tools of modern C++

Types and Initialization

init.cpp

```
int    count{0};
double price{9.99};
bool   ready{true};

int a = 3.7;     // compiles: a becomes 3 (data lost)
int b{3.7};      // error: narrowing conversion ✓ safer
```

- C plus plus has all the C types, plus a real bool type with true and false.
- Modern C plus plus prefers brace initialization. It works for every type, and it refuses …

2.01

auto

auto.cpp

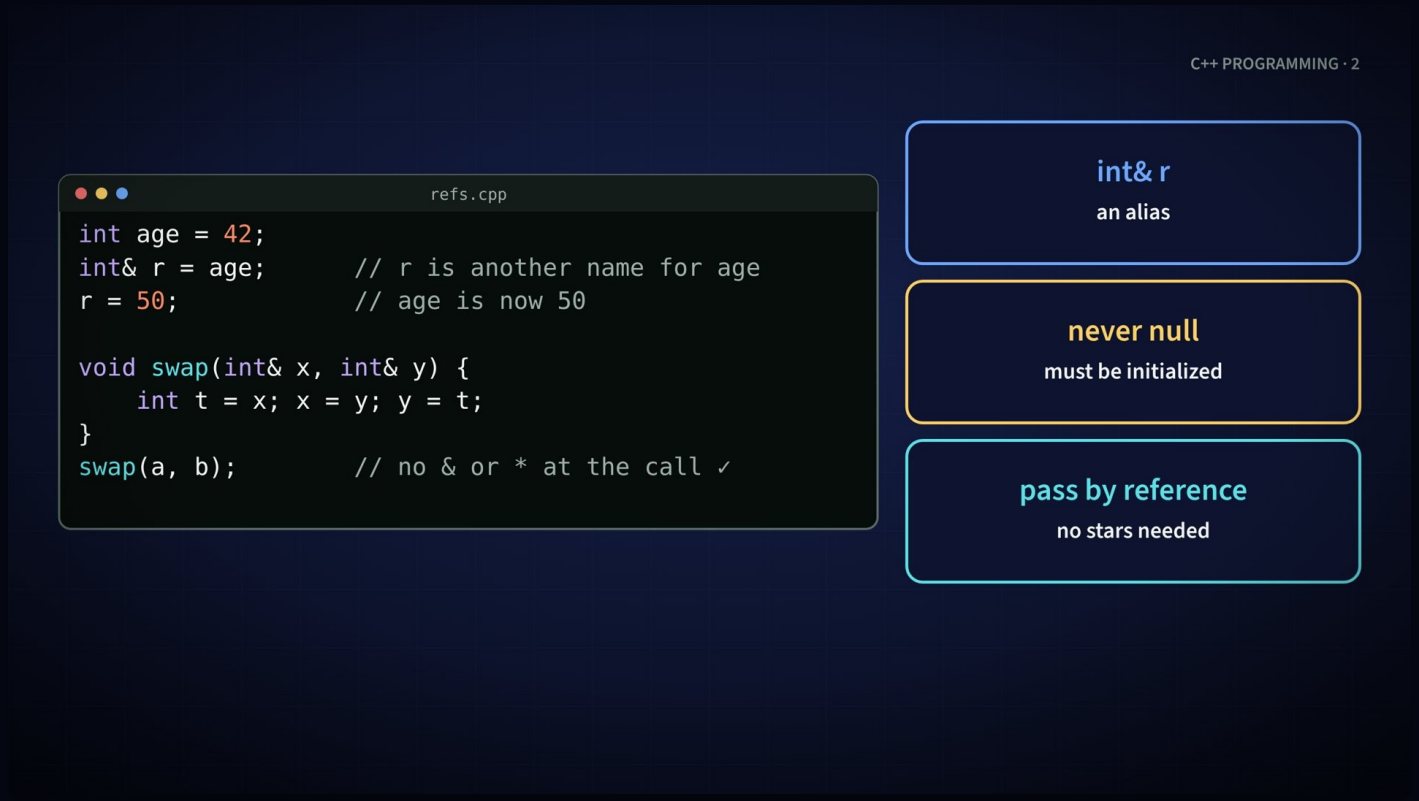
```cpp
auto n = 42;            // int
auto pi = 3.14159;      // double
auto name = std::string{"Kabul"};

std::map<std::string, std::vector<int>> scores;
auto it = scores.begin();   // instead of a very long type
```

- auto lets the compiler deduce the type from the value. It saves typing for long type names, …
- Use it when the type is obvious or unimportant. The variable still has one fixed type; C …

2.02

References



- int& r — an alias
- never null — must be initialized
- pass by reference — no stars needed

2.03

const and constexpr

const.cpp

```cpp
void print_report(const std::vector<Student>& list);  // no copy, no change

constexpr int square(int x) { return x * x; }
constexpr int AREA = square(12);    // 144, computed by the compiler
```

- const means a value cannot change. Passing large objects as a const reference avoids a copy …
- constexpr goes further: the value is computed at compile time, so it costs nothing when the …

2.04

std::string



- + — join
- == — compare text
- .size() .find() — length, search

2.05

Namespaces

namespaces.cpp

```cpp
namespace school {
    double average(const std::vector<int>& v);
}
namespace finance {
    double average(const std::vector<double>& v);
}

school::average(grades);       // no collision

using std::cout;               // OK: one name
using namespace std;           // ✗ avoid in headers
```

- A namespace groups names to avoid collisions. Two libraries can both have a function called …
- The standard library lives in namespace std. Writing using namespace std at the top of a …

2.06

Range-Based for

range_for.cpp

```cpp
std::vector<int> scores{90, 85, 77};

for (const auto& s : scores) {     // read, no copies
    std::cout << s << " ";
}
for (auto& s : scores) {           // change in place
    s += 5;
}
```

- Modern C plus plus has a simpler for loop that visits every element of a container. You do …
- Use a const reference to read elements without copying, and a plain reference to change them.

2.07



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 3

Functions, Overloading and Lambdas

Default arguments, inline functions and lambda expressions

Function Overloading

overload.cpp

```
int    area(int side)              { return side * side; }
int    area(int w, int h)          { return w * h; }
double area(double radius)         { return 3.14159 * radius * radius; }

area(4);        // 16      → square
area(3, 5);     // 15      → rectangle
area(2.0);      // 12.566  → circle
```

- In C, every function needs a unique name. In C plus plus, several functions can share a …
- The compiler picks the right version from the arguments you pass. It is the first, …

3.01

Default Arguments

defaults.cpp

```cpp
void greet(const std::string& name, const std::string& greeting = "Hello") {
    std::cout << greeting << ", " << name << "!\n";
}

greet("Mariam");            // Hello, Mariam!
greet("Ahmad", "Salaam");   // Salaam, Ahmad!
```

- A parameter can have a default value, used when the caller leaves it out. Default …
- This often replaces several overloads with one clear function.

3.02

Choosing How to Pass

Parameter	When
`int x`	small values: copy is cheap
`const std::string& s`	big objects you only read
`std::vector<int>& v`	the function changes it
`Node* p`	optional, may be nullptr

- Here is a simple rule. Pass small types like int and double by value. Pass large objects ...
- Pass by non-const reference when the function must change the caller's object. Pointers are ...

3.03

inline Functions

math_utils.h

```
#pragma once

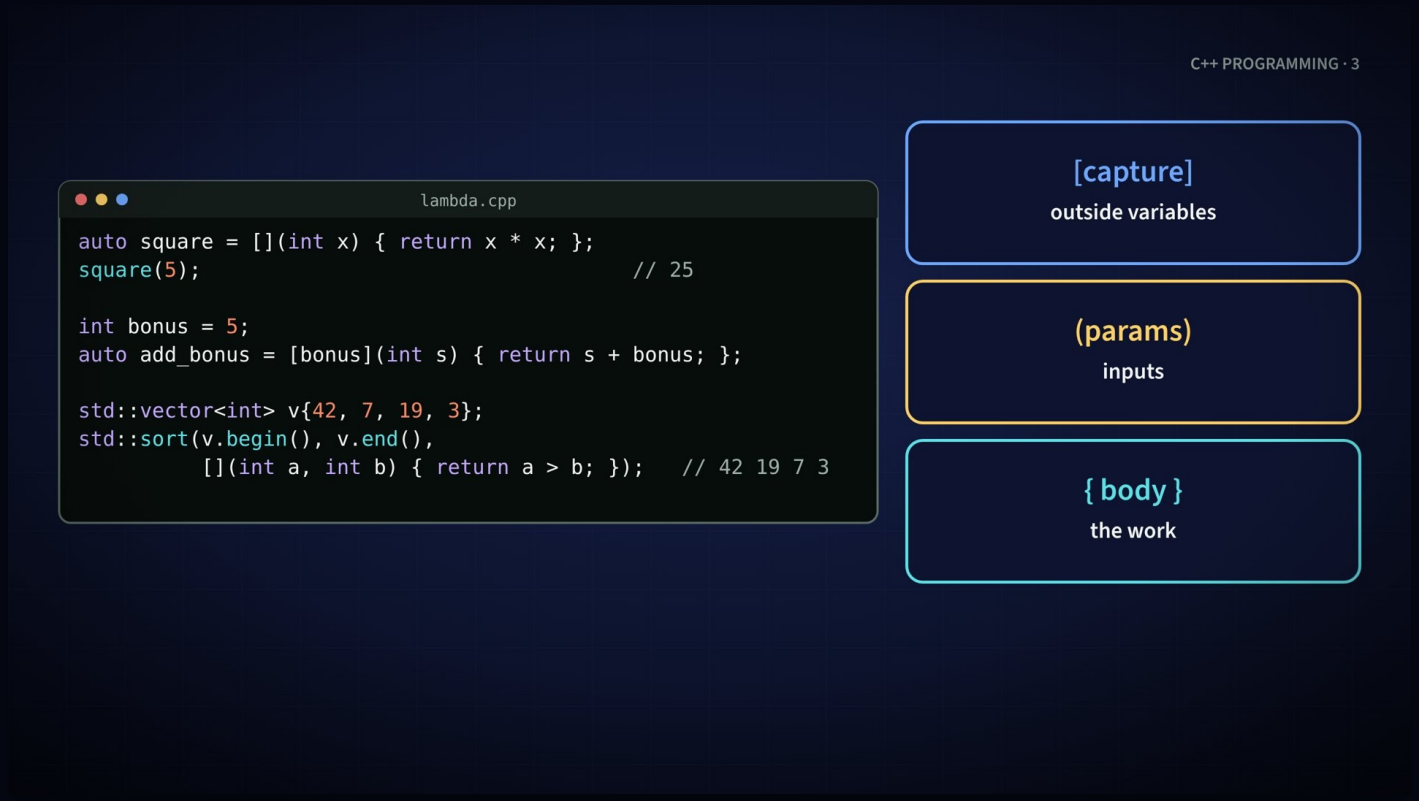
inline int clamp(int v, int lo, int hi) {
    return v < lo ? lo : (v > hi ? hi : v);
}

// safer than the C macro: real types, arguments evaluated once
```

- Calling a function has a small cost. For tiny functions, the compiler can paste the body …
- Modern compilers decide this themselves when optimizing. Today the inline keyword mainly …

3.04

Lambda Expressions



- [capture] — outside variables
- (params) — inputs
- { body } — the work

3.05

Capturing by Value and by Reference

capture.cpp

```cpp
int count = 0;

auto by_value = [count]()  { return count + 1; };   // copy
auto by_ref   = [&count]() { count++; };            // original

by_ref(); by_ref();
std::cout << count;     // 2
```

- Capturing by value copies the variable into the lambda. Capturing with an ampersand uses …
- Be careful: a lambda that captures by reference must not outlive the variables it refers to.

3.06



A FREE MYWEBUNIVERSITY COURSE

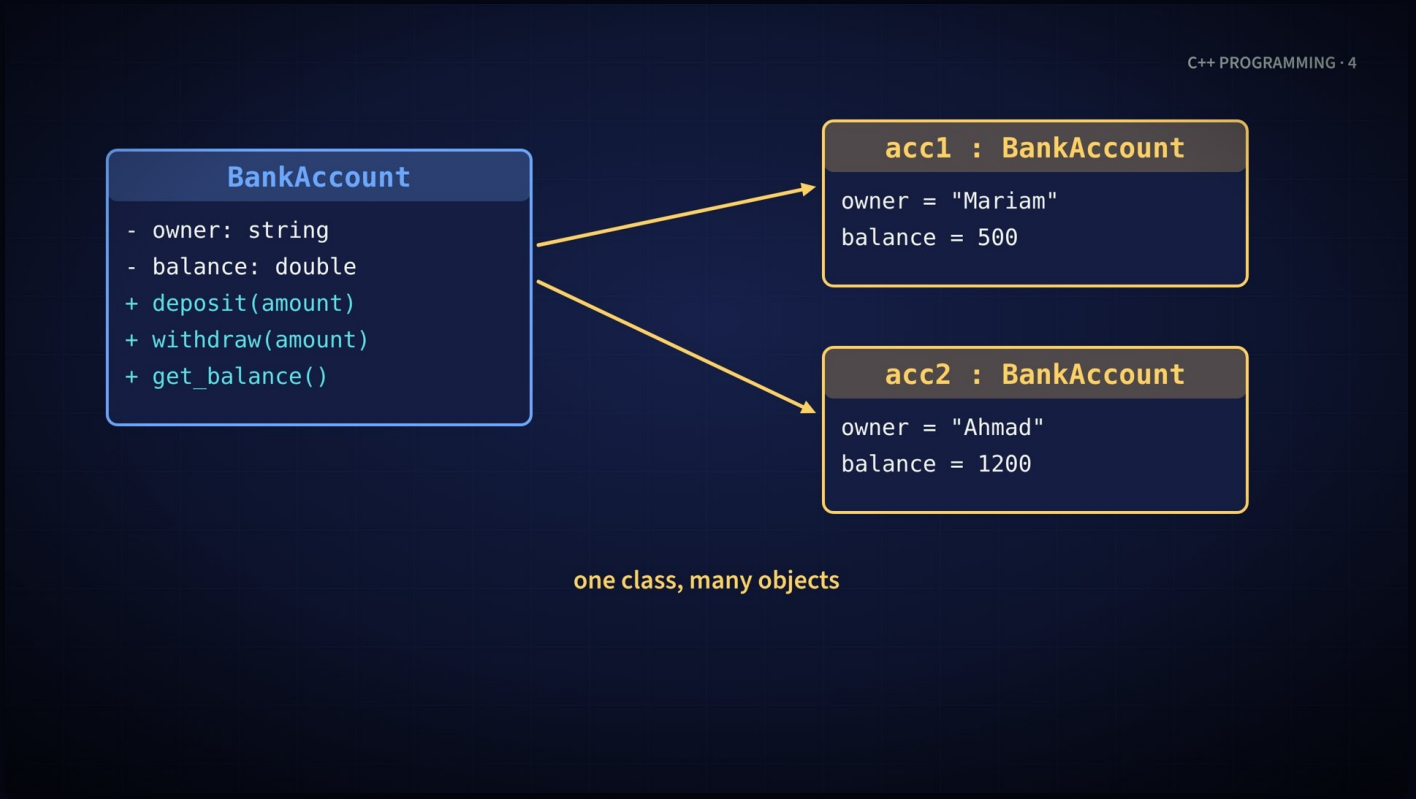
C++ PROGRAMMING

LESSON 4

Classes and Objects

Bundling data and behavior together

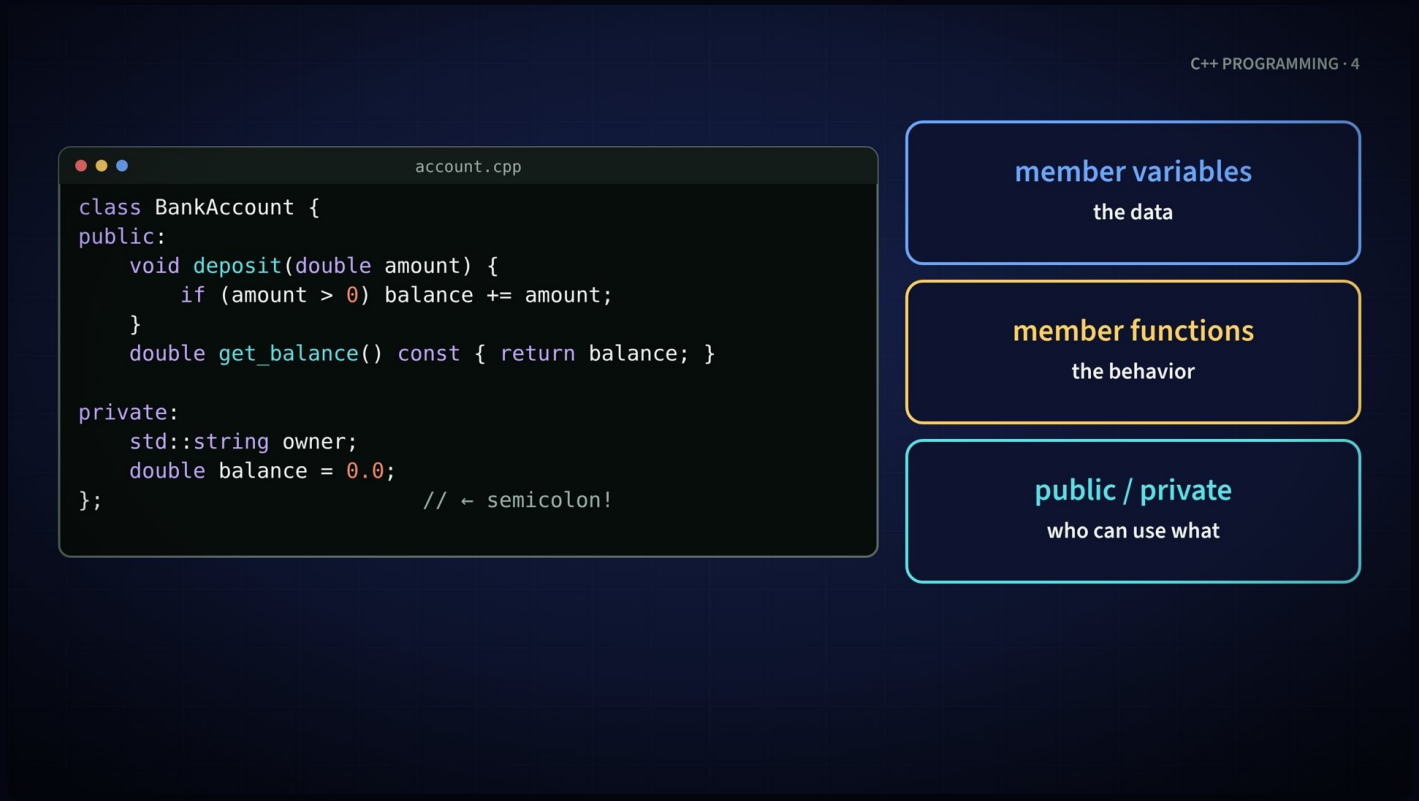
Class: A Blueprint



- A class is a blueprint that describes a kind of thing: what data it holds and what it can …
- One BankAccount class can create thousands of account objects, each with its own owner and …

4.01

Defining a Class



- member variables — the data
- member functions — the behavior
- public / private — who can use what

4.02

Creating and Using Objects

main.cpp

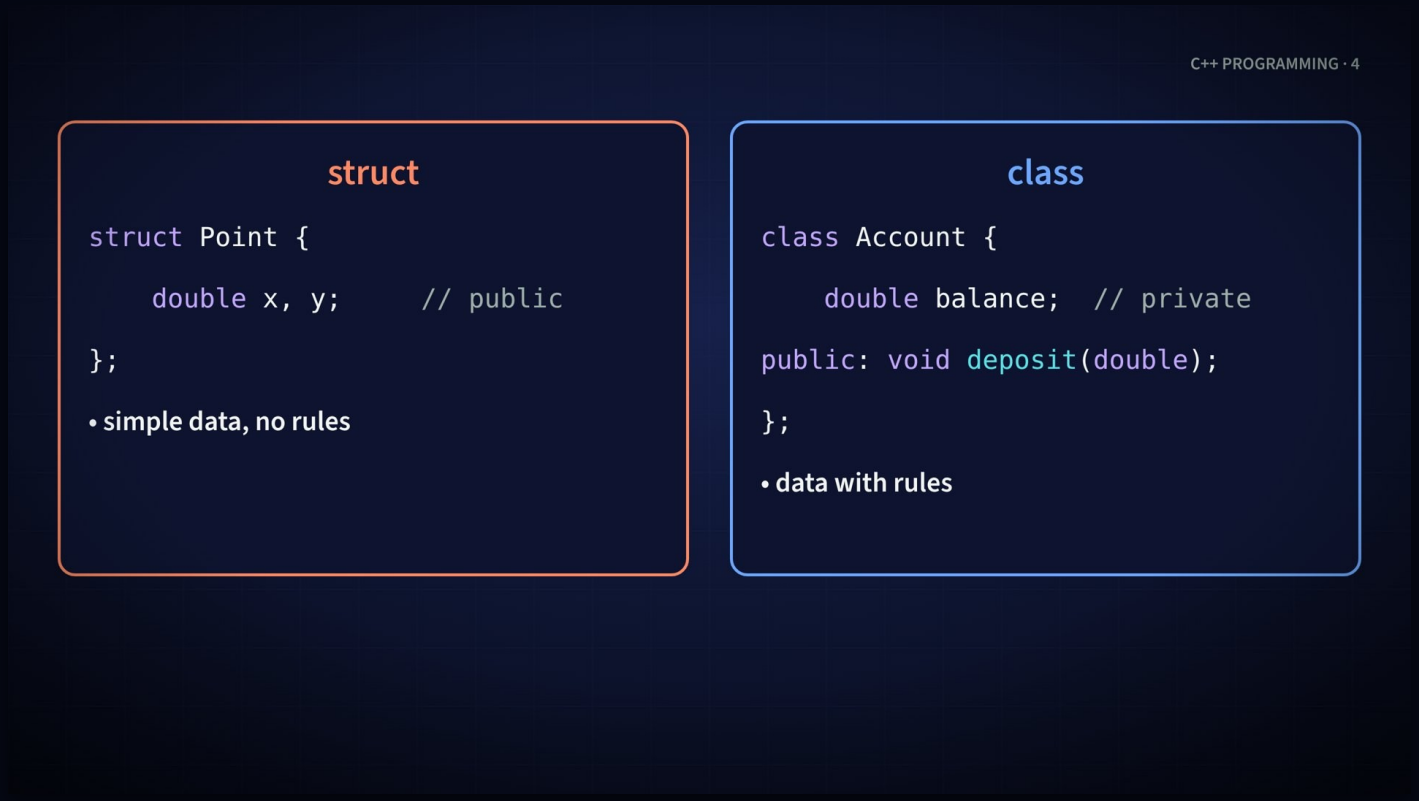
```
BankAccount acc;
acc.deposit(500);
acc.deposit(-90);              // ignored by the class rule
std::cout << acc.get_balance();  // 500

acc.balance = -1000;   // ✗ error: 'balance' is private
```

- Creating an object is like declaring a variable whose type is the class. Call methods with …
- Trying to touch a private member from outside is a compile error. The class protects its …

4.03

class vs struct



- In C plus plus, struct and class are almost the same. The only difference: struct members …
- By convention, use struct for simple data bundles with no rules, and class when the object …

4.04

The this Pointer and const Methods

this.cpp

```
class Counter {
public:
    Counter& add(int n) { this->value += n; return *this; }
    int get() const { return value; }      // const: read only
private:
    int value = 0;
};

Counter c;
c.add(2).add(3);          // chaining via *this
std::cout << c.get();     // 5
```

- Inside a member function, this is a pointer to the object the function was called on. It is …
- A method marked const promises not to change the object. Getters should be const, so they …

4.05

Header and Source Files

account.hpp · account.cpp

```
// account.hpp
#pragma once
class BankAccount {
public:
    void deposit(double amount);
private:
    double balance = 0.0;
};

// account.cpp
#include "account.hpp"
void BankAccount::deposit(double amount) {
    if (amount > 0) balance += amount;
}
```

- Real projects split a class in two. The header, with a .h or .hpp extension, holds the …
- The source file, ending in .cpp, holds the method bodies, written with the class name and …

4.06



A FREE MYWEBUNIVERSITY COURSE

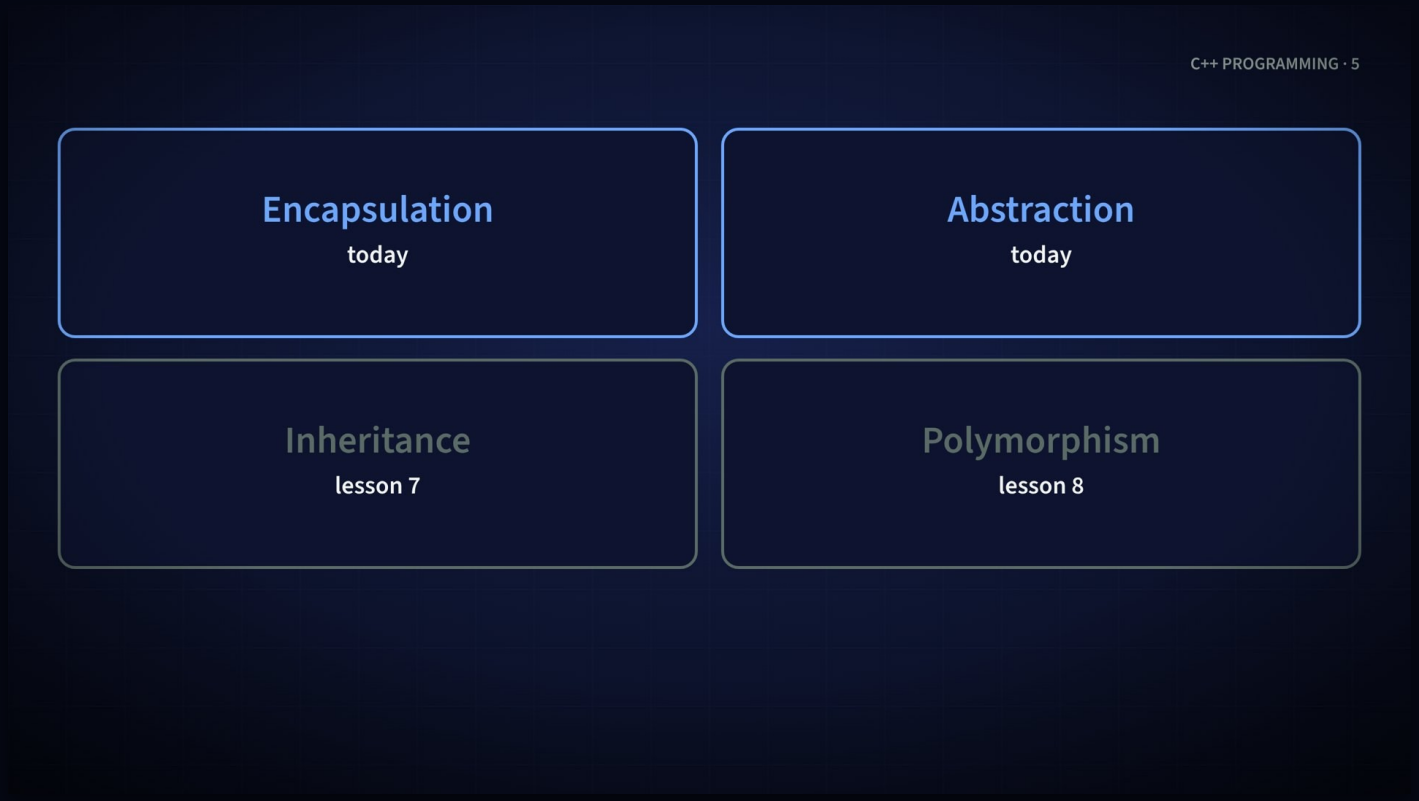
C++ PROGRAMMING

LESSON 5

Encapsulation, Data Hiding and Abstraction

Two of the four pillars of object-oriented programming

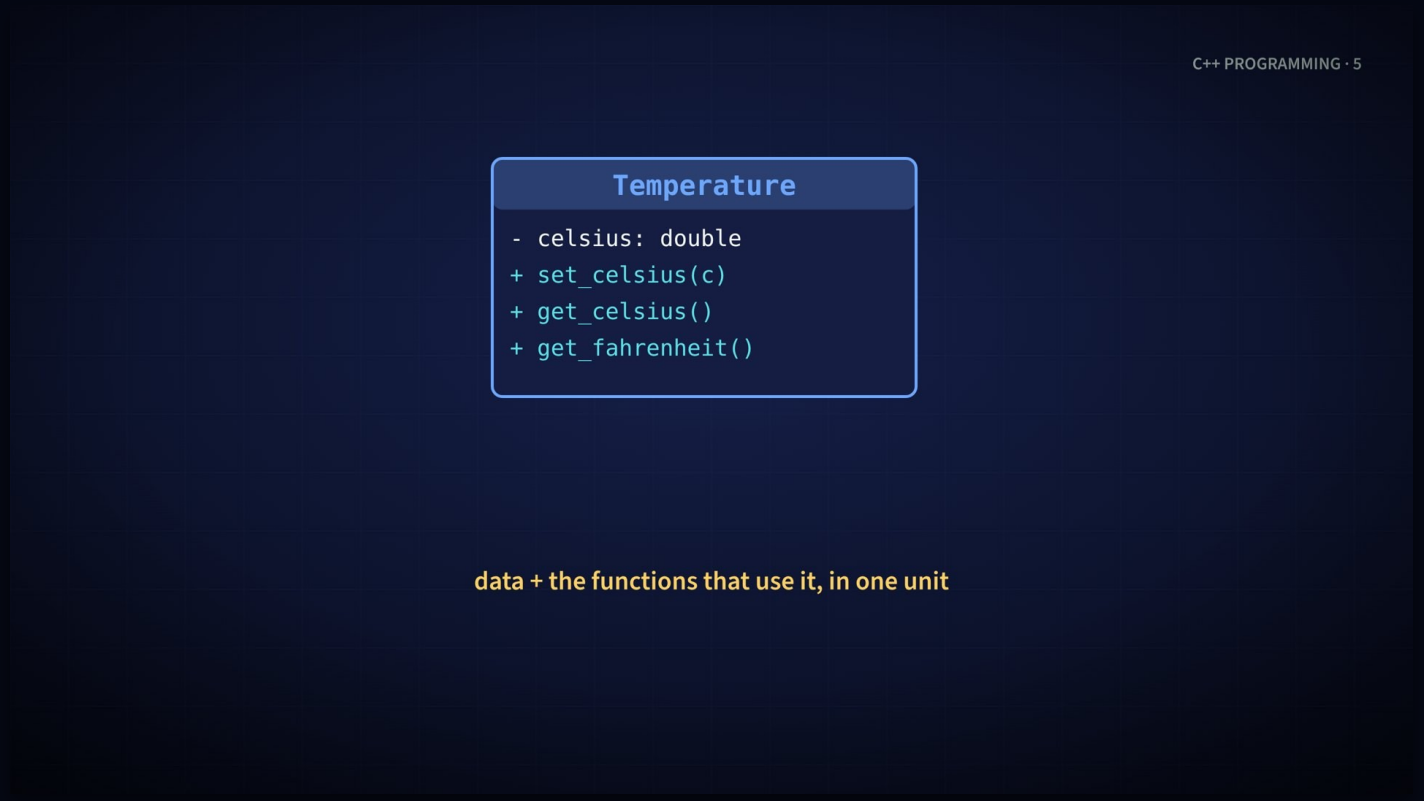
The Four Pillars



- Encapsulation — today
- Abstraction — today
- Inheritance — lesson 7

5.01

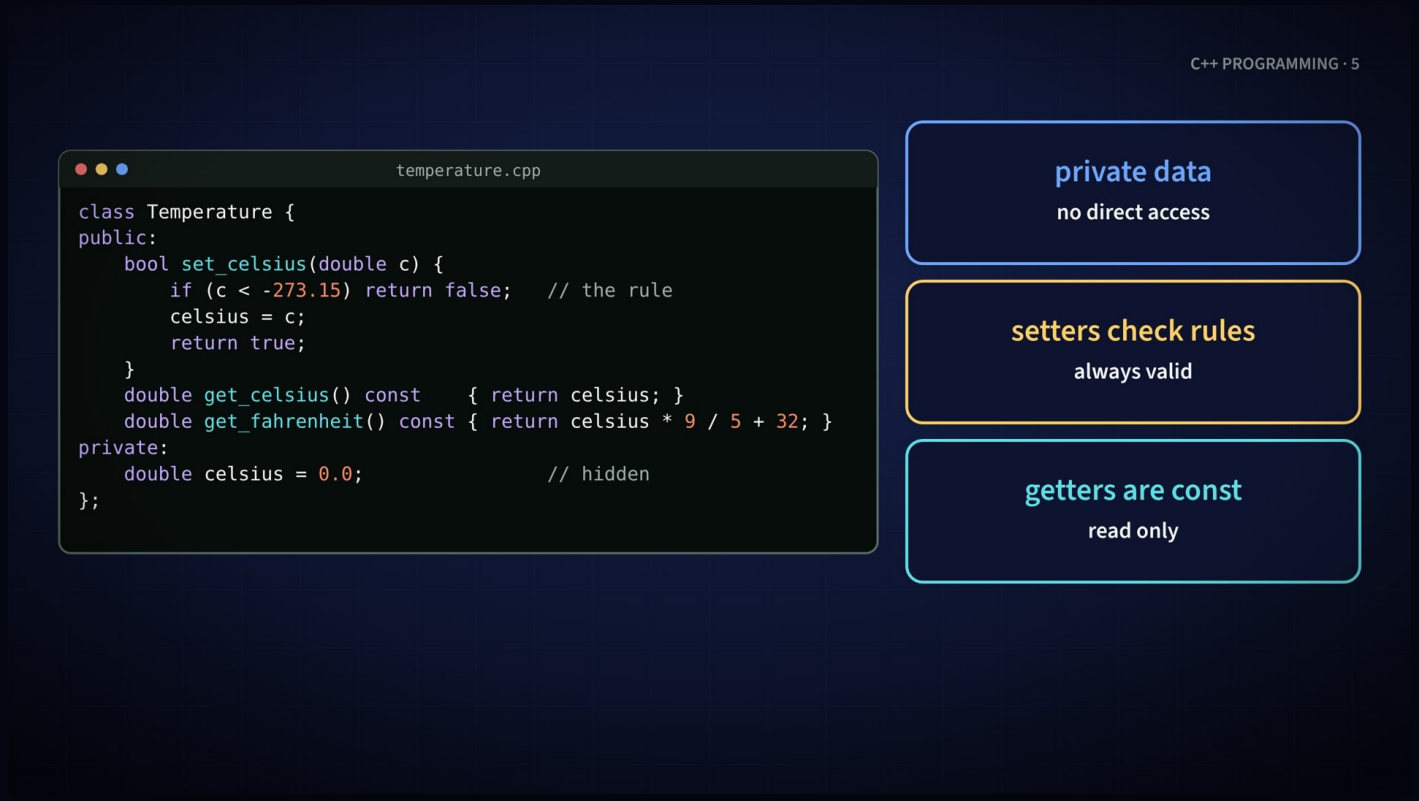
Encapsulation



- Encapsulation means bundling data together with the functions that work on it, inside one …
- The outside world talks to the object only through its public methods. The object decides …

5.02

Data Hiding with private



- private data — no direct access
- setters check rules — always valid
- getters are const — read only

5.03

public, private and protected

Access	The class	Derived class	Everyone
public	✓	✓	✓
protected	✓	✓	✗
private	✓	✗	✗

- C plus plus has three access levels. public members can be used by anyone. private members …
- protected members are like private, but derived classes can use them too; we will need that …

5.04

Abstraction



- Abstraction means showing only what the user needs, and hiding how it works. When you drive ...
- std vector is a perfect example. You call push back and size, and never think about the ...

5.05

Abstract Interfaces

storage.hpp

```cpp
class Storage {                          // an interface
public:
    virtual ~Storage() = default;
    virtual void save(const std::string& key, const std::string& value) = 0;
    virtual std::string load(const std::string& key) = 0;
};

class FileStorage  : public Storage { /* saves to files    */ };
class CloudStorage : public Storage { /* saves to a server */ };
```

- You can describe an abstraction as an interface: a class with only pure virtual functions, …
- Code that uses the interface works with any class that implements it. We will see exactly …

5.06

Why It Matters



- Fewer bugs — data always valid
- Easy to change — inside can be rewritten
- Teamwork — clear interfaces

5.07



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 6

Constructors, Destructors and RAII

How objects are born, copied, moved and destroyed

Constructors



- same name as class — no return type
- : member{value} — initializer list

6.01

Destructors

lifetime.cpp

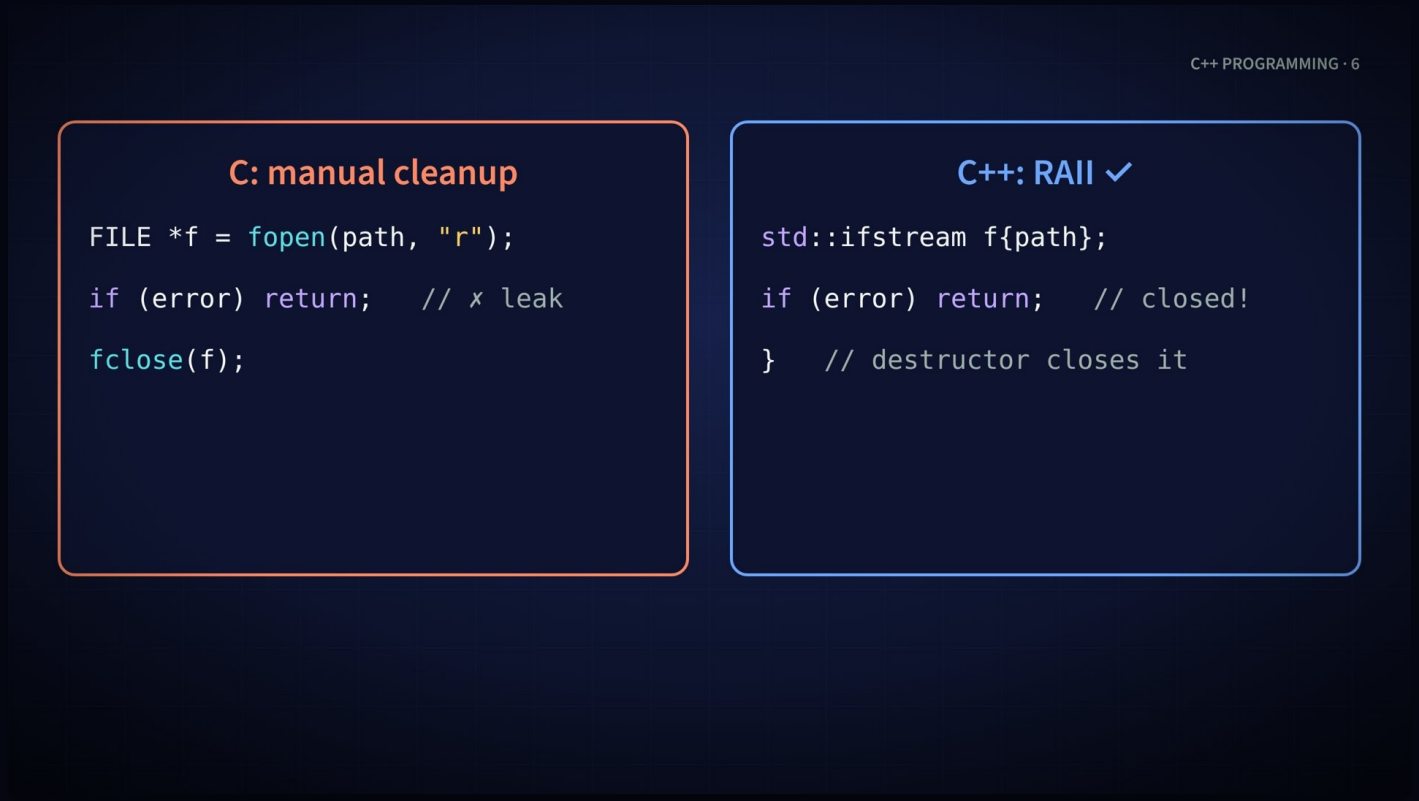
```
class Logger {
public:
    Logger()  { std::cout << "start\n"; }
    ~Logger() { std::cout << "end\n"; }       // destructor
};

void work() {
    Logger log;       // prints: start
    /* ... */
}                     // prints: end  (automatic!)
```

- A destructor runs automatically when an object is destroyed: when it goes out of scope, or …
- Its job is to clean up: close files, release memory, unlock locks. Objects are destroyed in …

6.02

RAII: The Most Important Idea



- RAII stands for Resource Acquisition Is Initialization. It means: get a resource in the …
- Because destructors run automatically, the resource is always released, even if the …
- This is how std string, std vector, file streams and smart pointers manage memory and files …

6.03

Copying Objects

copy.cpp

```
Student a{"Mariam", 1};
Student b = a;          // copy constructor
Student c{"Ahmad", 2};
c = a;                  // copy assignment

class Buffer {          // ✗ dangerous
    int* data = new int[100];
    ~Buffer() { delete[] data; }
};                      // default copy → double delete!
```

- When you copy an object, the copy constructor runs. When you assign one existing object to …
- For members like strings and vectors, the default copy does the right thing. But if your …

6.04

The Rule of Zero, Three and Five



- Rule of zero ✓ — use string, vector, smart pointers
- Rule of three — destructor + copy ctor + copy =
- Rule of five — + move ctor + move =

6.05

Move Semantics

move.cpp

```cpp
std::vector<int> big(10'000'000, 1);

std::vector<int> a = big;              // copy: 10 million ints
std::vector<int> b = std::move(big);   // move: just pointers
                                       // big is now empty
std::vector<int> make_data() {
    std::vector<int> v(1000);
    return v;                          // moved, not copied
}
```

- Copying a large vector copies every element. But when the source is about to disappear …
- std move says you no longer need an object, so it may be moved from. Returning local …

6.06



A FREE MYWEBUNIVERSITY COURSE

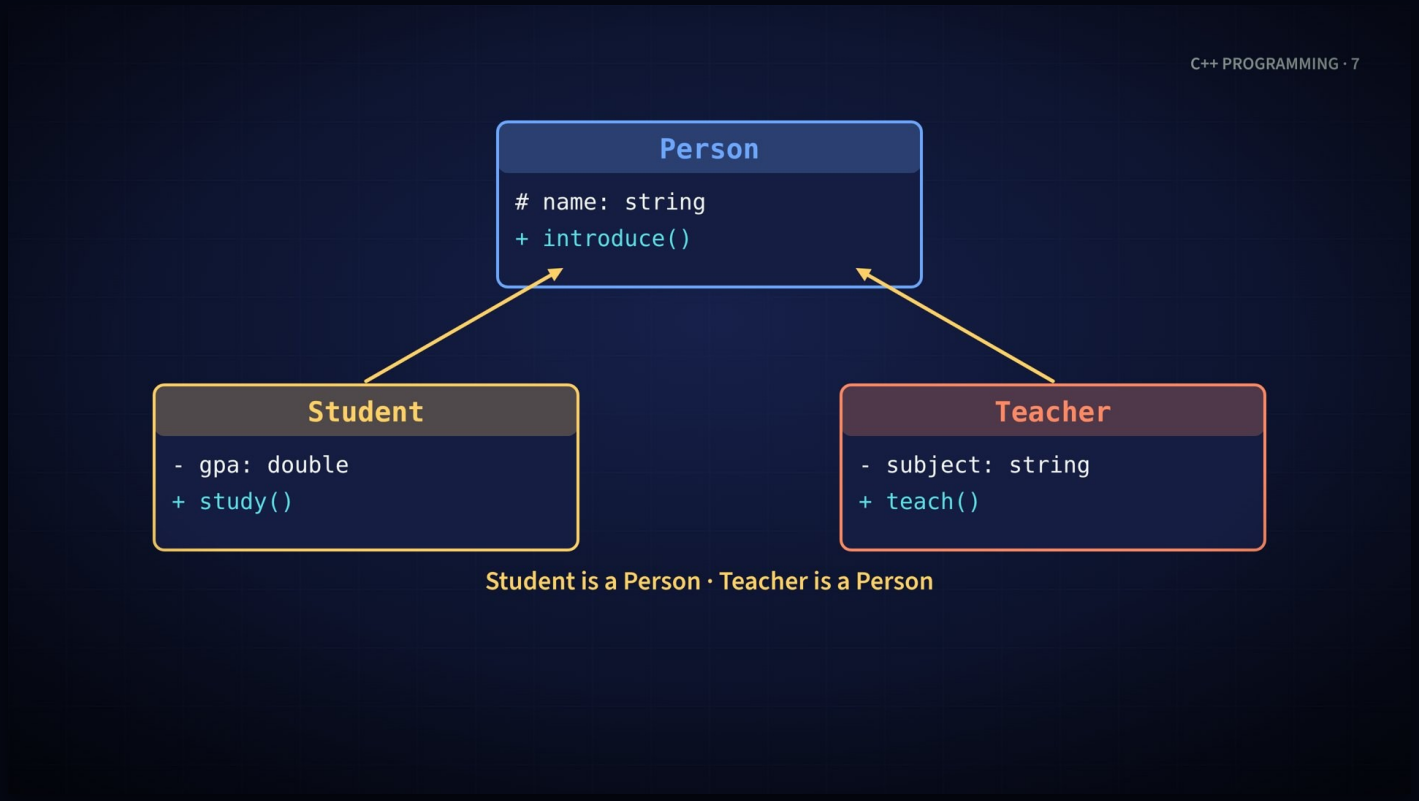
C++ PROGRAMMING

LESSON 7

Inheritance

Reusing and extending classes

Is-a Relationships



- Inheritance lets a new class, called the derived class, reuse everything from an existing ...
- Use inheritance for an is-a relationship. A savings account is a bank account. A circle is ...

7.01

Deriving a Class

person.cpp

```cpp
class Person {
public:
    Person(std::string n) : name{std::move(n)} {}
    void introduce() const { std::cout << "I am " << name << "\n"; }
protected:
    std::string name;
};

class Student : public Person {
public:
    Student(std::string n, double g) : Person{std::move(n)}, gpa{g} {}
    void study() const { std::cout << name << " is studying\n"; }
private:
    double gpa;
};
```

- Write a colon and public after the class name, then the base class. The derived class gets …
- Its constructor should first call the base constructor, in the initializer list, so the …

7.02

Using Derived Objects

main.cpp

```
Student s{"Mariam", 3.9};
s.introduce();     // inherited from Person: I am Mariam
s.study();         // added by Student:      Mariam is studying

// construction:  Person → Student
// destruction:   ~Student → ~Person
```

- A Student object can call introduce, which it inherited, and study, which it added.
- When a Student is created, the Person part is constructed first, then the Student part. …

7.03

Overriding Behavior

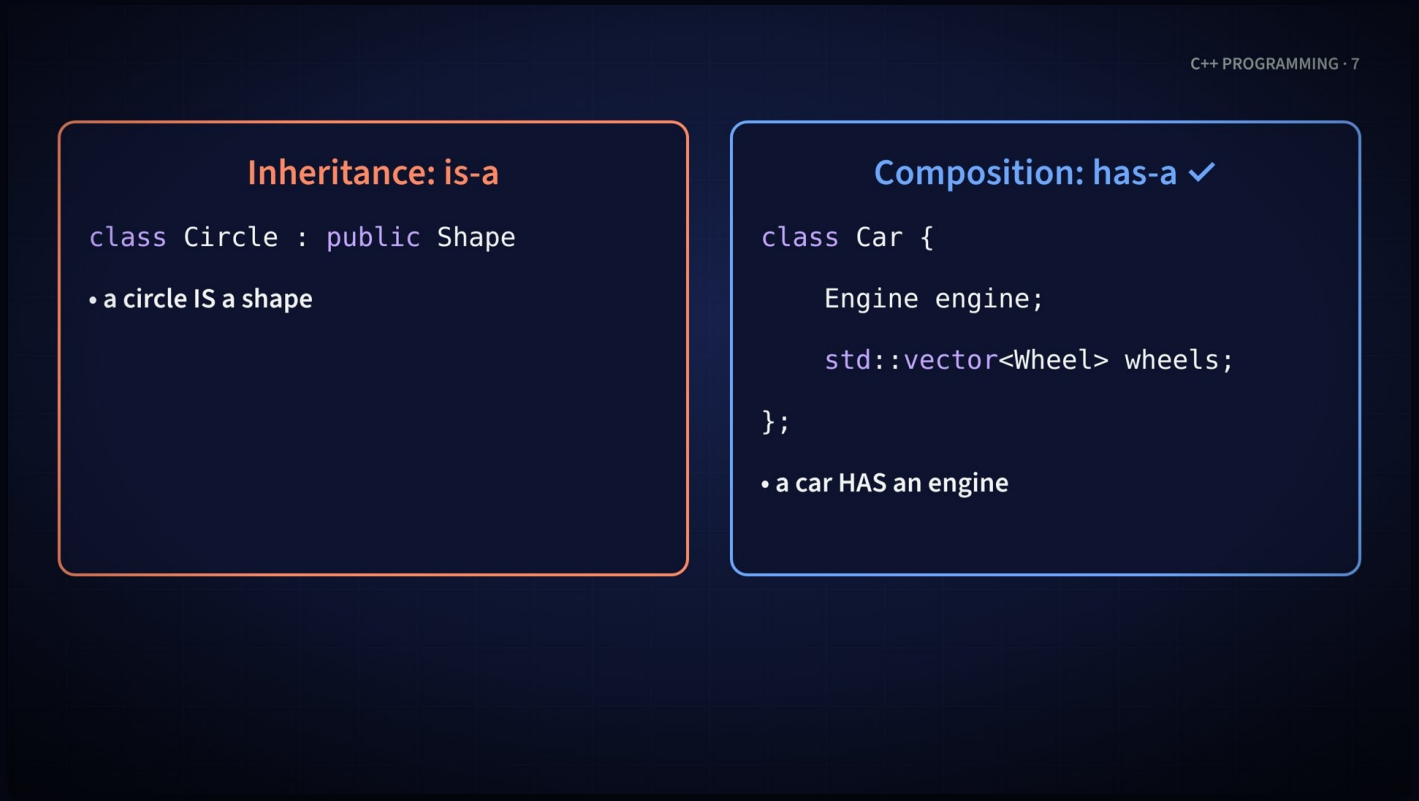
override.cpp

```cpp
class Person {
public:
    virtual void introduce() const { std::cout << "I am " << name << "\n"; }
    virtual ~Person() = default;
protected:
    std::string name;
};
class Teacher : public Person {
public:
    void introduce() const override {
        std::cout << "I teach " << subject << "\n";
    }
private:
    std::string subject;
};
```

- A derived class can replace a base class function with its own version. Mark the base …
- override asks the compiler to check that you really are replacing a base function. A typing …

7.04

Is-a vs Has-a



- Not every relationship should use inheritance. A car has an engine; it is not an engine. …
- A good rule: prefer composition, and use inheritance only when the is-a test truly passes.

7.05

Multiple Inheritance

multiple.cpp

```
class Printable { public: virtual void print() const = 0; virtual ~Printable() = default; };
class Saveable  { public: virtual void save()  const = 0; virtual ~Saveable()  = default; };

class Report : public Document, public Printable, public Saveable {
    void print() const override { /* ... */ }
    void save()  const override { /* ... */ }
};
```

- C plus plus allows a class to inherit from more than one base class. It is powerful but can …
- The safe, common use is inheriting from one real base class plus small interfaces that …

7.06



A FREE MYWEBUNIVERSITY COURSE

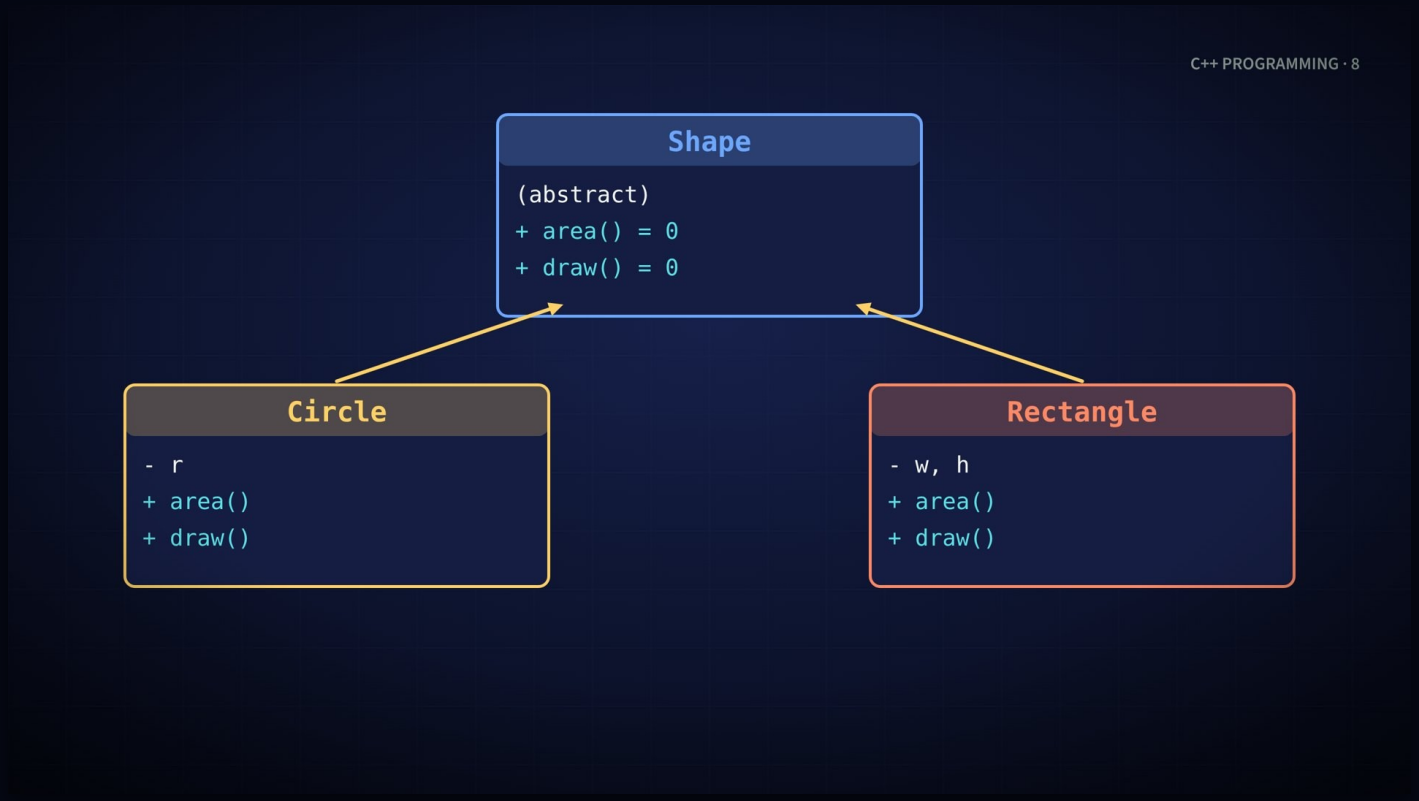
C++ PROGRAMMING

LESSON 8

Polymorphism and Virtual Functions

One interface, many forms

Many Forms



- Polymorphism means many forms. One piece of code can work with objects of many different …
- A drawing program keeps a list of shapes and simply calls draw on each one. Circles draw …

8.01

virtual Functions

shapes.cpp

```cpp
class Shape {
public:
    virtual double area() const = 0;        // pure virtual
    virtual ~Shape() = default;
};
class Circle : public Shape {
    double r;
public:
    explicit Circle(double r) : r{r} {}
    double area() const override { return 3.14159 * r * r; }
};
class Rect : public Shape {
    double w, h;
public:
    Rect(double w, double h) : w{w}, h{h} {}
    double area() const override { return w * h; }
};
```

- Mark a base class function virtual to allow derived classes to override it. Then, when you …
- Without virtual, the base version would always run. This is called dynamic dispatch, or …

8.02

One Loop, Many Shapes

main.cpp

```cpp
std::vector<std::unique_ptr<Shape>> shapes;
shapes.push_back(std::make_unique<Circle>(1.0));
shapes.push_back(std::make_unique<Rect>(2.0, 3.0));

for (const auto& s : shapes) {
    std::cout << s->area() << "\n";   // 3.14159, then 6
}
```

- Now we store different shapes in one vector of smart pointers to Shape, and call area on …
- The loop does not know or care which shape it is. Add a Triangle class tomorrow, and this …

8.03

Pure Virtual and Abstract Classes

abstract.cpp

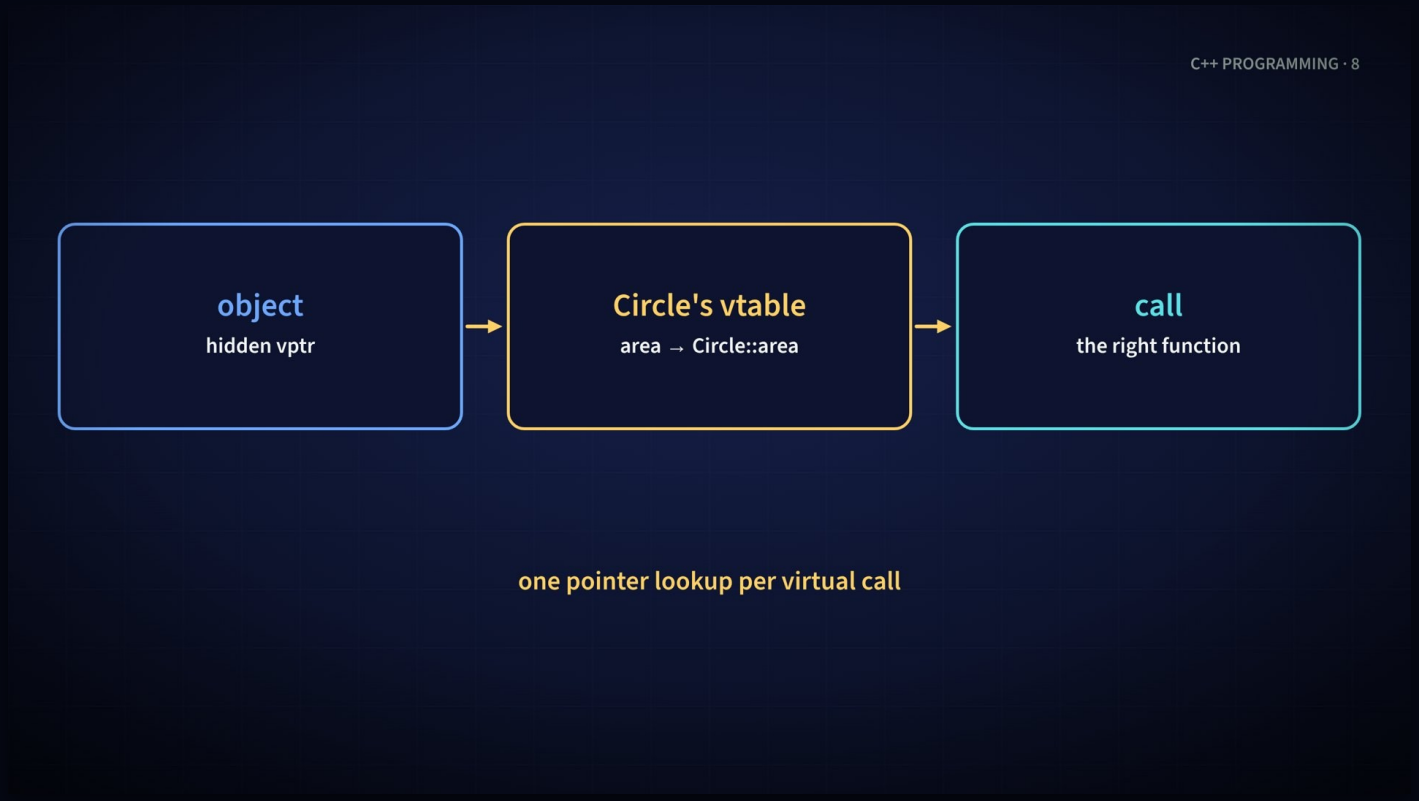
```
Shape s;                    // ✗ error: Shape is abstract
Circle c{2.0};              // ✓ Circle implements area()

Shape& ref = c;             // ✓ base reference to a derived object
ref.area();                 // calls Circle::area()
```

- A pure virtual function, marked equals zero, has no body in the base class. A class with at …
- An abstract class is a contract. Every concrete derived class must implement all the pure …

8.04

How It Works: the vtable



- object — hidden vptr
- Circle's vtable — area → Circle::area
- call — the right function

8.05

Virtual Destructors

vdtor.cpp

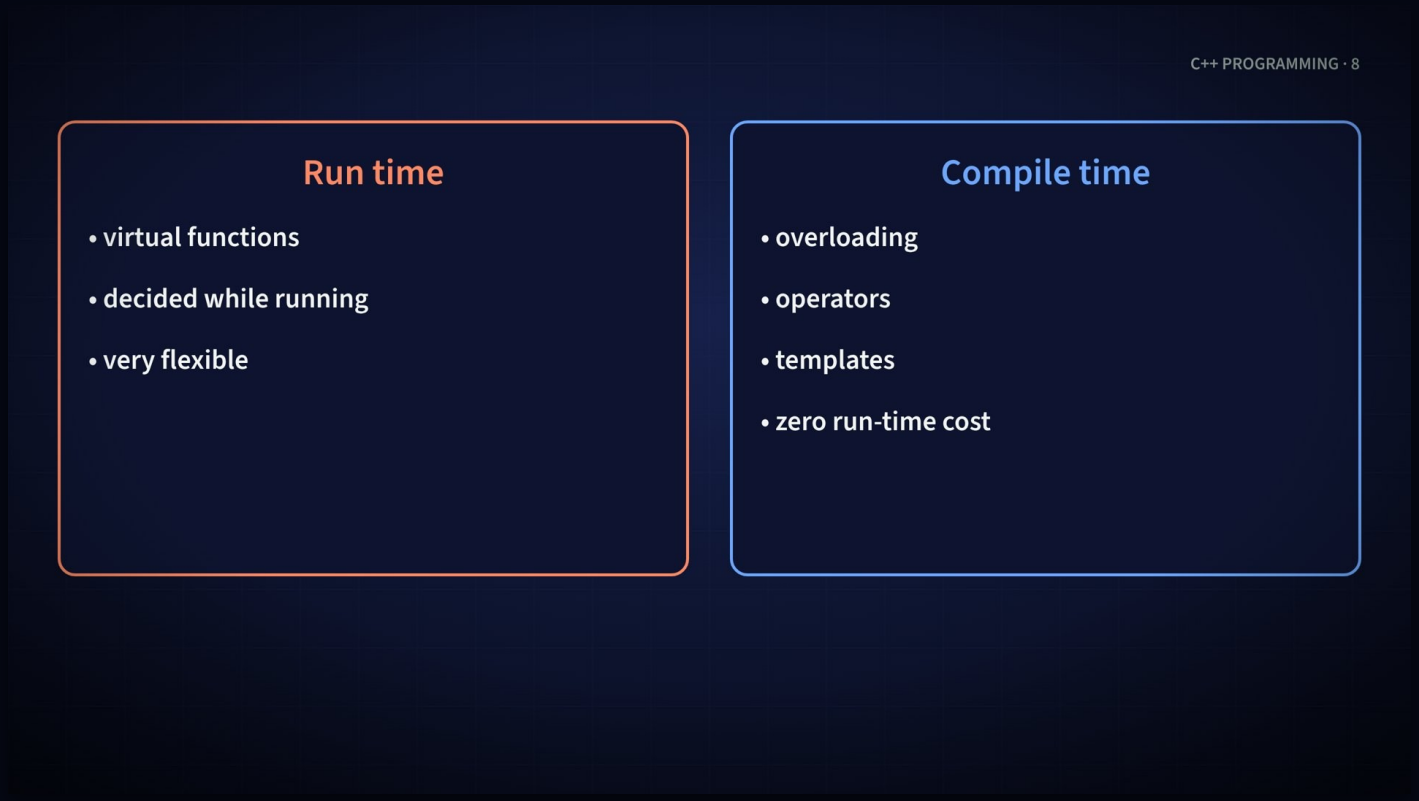
```
class Shape {
public:
    virtual ~Shape() = default;     // ✓ required for polymorphism
};

Shape* p = new Circle{1.0};
delete p;                           // runs ~Circle, then ~Shape
```

- If you delete a derived object through a base pointer, the base class destructor must be …
- The rule is simple: any class meant to be used polymorphically needs a virtual destructor.

8.06

Compile-Time Polymorphism



- There are two kinds of polymorphism. Run-time polymorphism uses virtual functions, decided ...
- Compile-time polymorphism is decided by the compiler: function overloading, operator ...

8.07



A FREE MYWEBUNIVERSITY COURSE

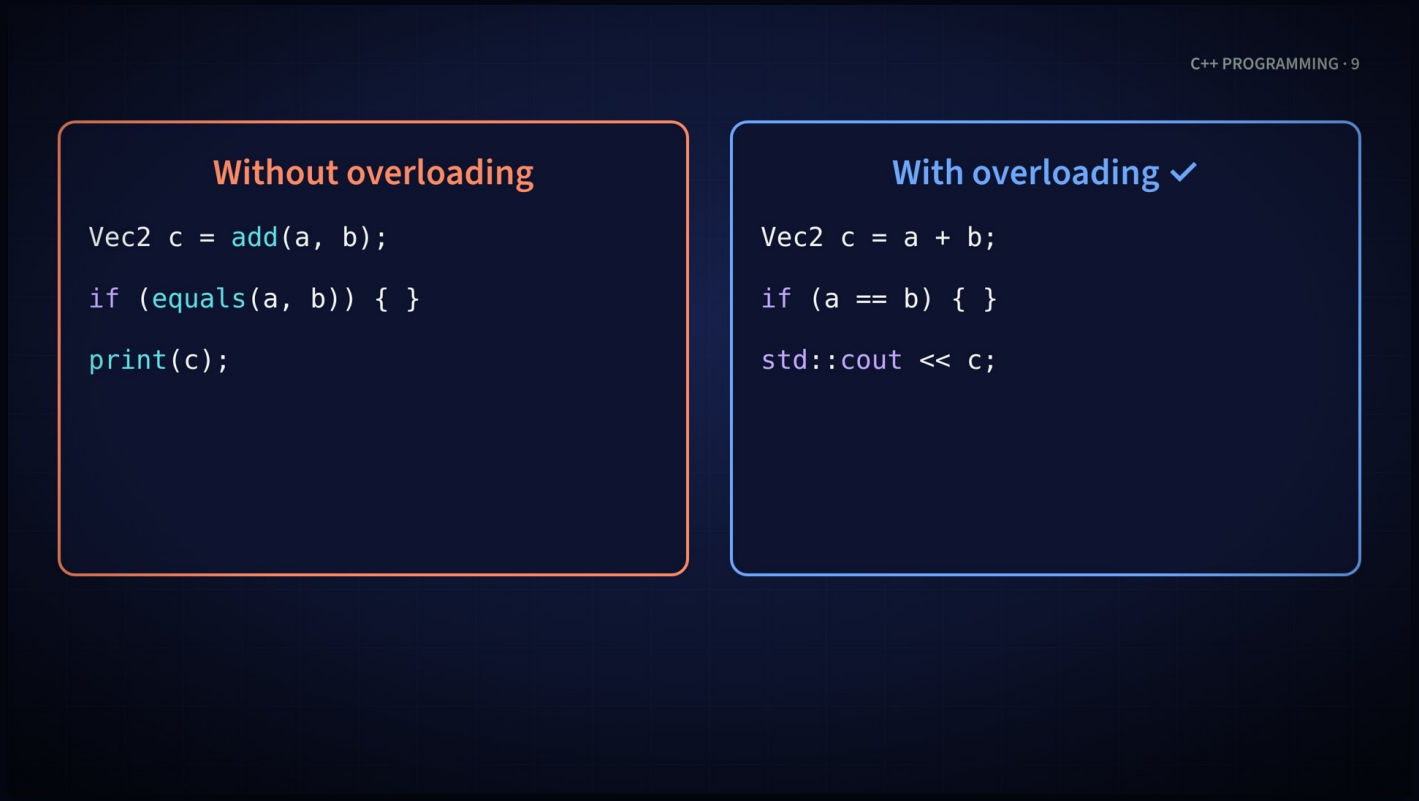
C++ PROGRAMMING

LESSON 9

Operator Overloading

Making your own types feel built in

Why Overload Operators?



- Built-in types use natural operators: a plus b. With operator overloading, your own types …
- This is compile-time polymorphism: the same operator works on many types. std string …

9.01

Overloading + and +=

vec2.cpp

```
struct Vec2 {
    double x = 0, y = 0;

    Vec2& operator+=(const Vec2& o) {
        x += o.x; y += o.y;
        return *this;
    }
};

Vec2 operator+(Vec2 a, const Vec2& b) { return a += b; }

Vec2 p{1, 2}, q{3, 4};
Vec2 r = p + q;      // {4, 6}
```

- An operator is just a function with a special name: operator, followed by the symbol.
- A common pattern is to write plus equals as a member function that changes the object, and …

9.02

Comparison Operators

compare.cpp

```cpp
struct Version {
    int major, minor, patch;
    auto operator<=>(const Version&) const = default;   // C++20
};

Version a{1, 4, 2}, b{1, 10, 0};
a <  b;    // true
a == b;    // false
a >= b;    // false
```

- Define double equals to compare objects by value. Since C plus plus 20, you can ask the …
- The three-way comparison operator, nicknamed the spaceship operator because of its shape, …

9.03

Printing with <<

print.cpp

```cpp
class Money {
public:
    explicit Money(long cents) : cents{cents} {}
    friend std::ostream& operator<<(std::ostream& os, const Money& m) {
        return os << m.cents / 100 << "." << std::setw(2) << std::setfill('0') << m.cents % 100;
    }
private:
    long cents;
};

std::cout << "Total: " << Money{12345} << "\n";   // Total: 123.45
```

- To print your object with cout, overload the output operator. It is a free function that …
- If it needs private members, declare it a friend inside the class.

9.04

Subscript and Call Operators

operators.cpp

```
class Grid {
public:
    int& operator()(int r, int c) { return cells[r * cols + c]; }
private:
    int cols = 10;
    std::vector<int> cells = std::vector<int>(100);
};

struct Multiplier {
    int k;
    int operator()(int x) const { return x * k; }
};
Multiplier times3{3};
times3(7);    // 21
```

- Overloading square brackets makes your class act like an array, which is how vector and map …
- Overloading the call operator, parentheses, makes an object you can call like a function, …

9.05

Rules of Good Taste



- Obvious meaning — + adds, == compares
- Stay consistent — == with !=, < with >
- Cannot overload — . :: ?: sizeof

9.06



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 10

Memory and Smart Pointers

new and delete, unique_ptr, shared_ptr and weak_ptr

new and delete

raw.cpp

```
Student* s = new Student{"Mariam", 1};   // constructor runs
delete s;                                // destructor runs

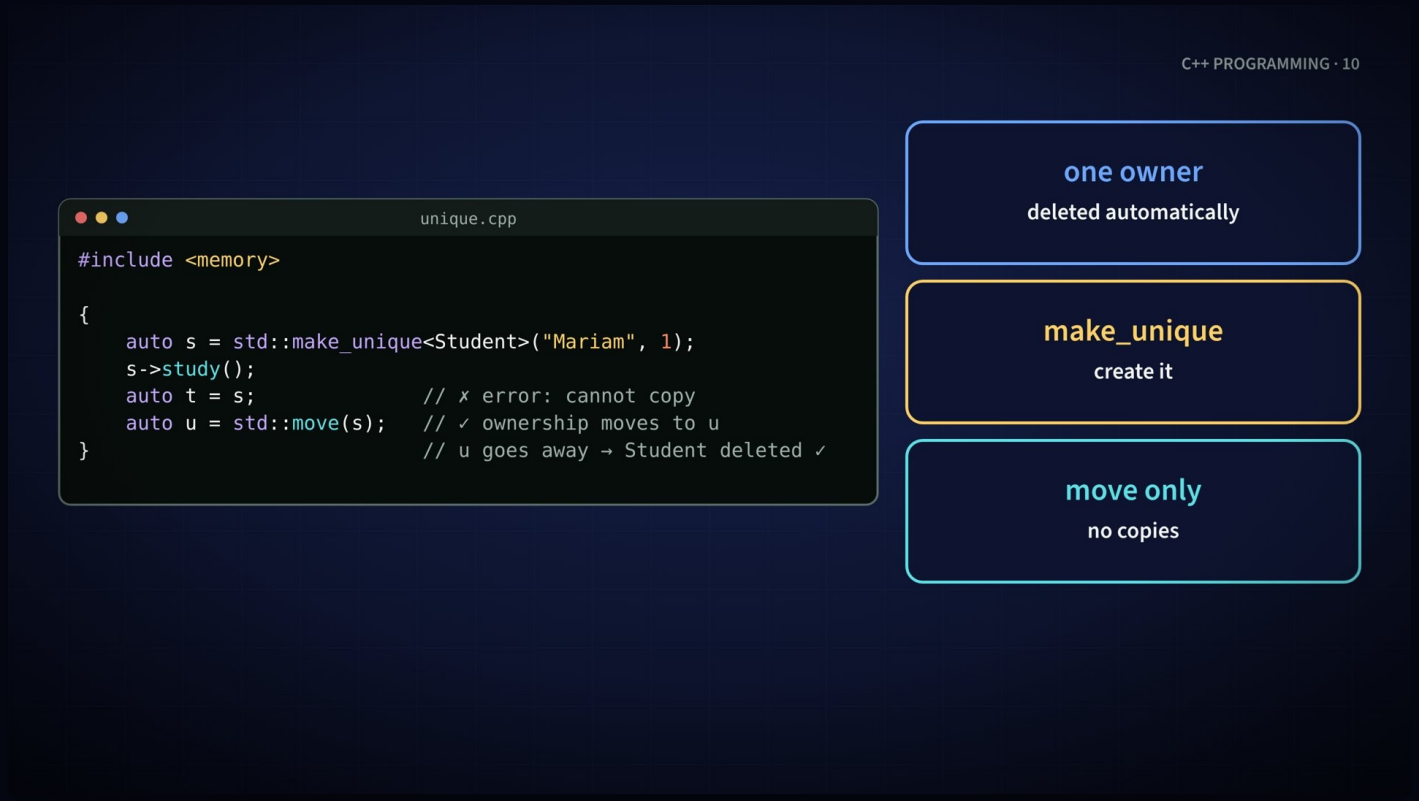
int* a = new int[100];
delete[] a;                              // [] for arrays

// ✗ same old dangers: forget delete → leak,
//   delete twice → crash, use after delete → bug
```

- C plus plus replaces malloc and free with new and delete. new allocates memory and runs the …
- Arrays use new with brackets and delete with brackets. But raw new and delete have every …

10.01

std::unique_ptr



- one owner — deleted automatically
- make_unique — create it
- move only — no copies

10.02

std::shared_ptr

shared.cpp

```
auto img = std::make_shared<Image>("map.png");   // count = 1
{
    auto viewer = img;                            // count = 2
    auto editor = img;                            // count = 3
}                                                 // count = 1
std::cout << img.use_count();                     // 1
img.reset();                                      // count = 0 → deleted
```

- Sometimes several parts of a program need the same object, and none of them knows who …
- Each copy increases the count, and each destruction decreases it. When the count reaches …

10.03

std::weak_ptr and Cycles

weak.cpp

```
struct Node {
    std::vector<std::shared_ptr<Node>> children;   // owns children
    std::weak_ptr<Node> parent;                    // observes parent
};

if (auto p = node->parent.lock()) {   // still alive?
    p->visit();
}
```

- If two objects hold shared pointers to each other, their counts never reach zero, and both …
- A weak pointer observes an object without owning it, so it does not increase the count. Use …

10.04

Which Pointer Should I Use?

Use	When
`T / std::vector<T>`	most of the time: no pointer
`std::unique_ptr<T>`	one owner (the default)
`std::shared_ptr<T>`	shared ownership
`std::weak_ptr<T>`	observe a shared object, break cycles
`T* / T&`	non-owning access only

- Here is the modern guideline. Prefer plain objects and containers; most of the time you …
- When you need a heap object with one owner, use unique pointer. For truly shared ownership, …

10.05

C vs C++ Memory

C

```c
int *a = malloc(n * sizeof *a);
if (!a) return -1;
/* every path must free(a) */
free(a);
```

Modern C++

```cpp
std::vector<int> a(n);
auto p = std::make_unique<Big>();
// nothing to free
// destructors clean up
```

- Compare the two worlds. In C, every malloc needs a free on every path, and tools must catch …
- In modern C plus plus, ownership is written in the types, and destructors clean up …

10.06



A FREE MYWEBUNIVERSITY COURSE

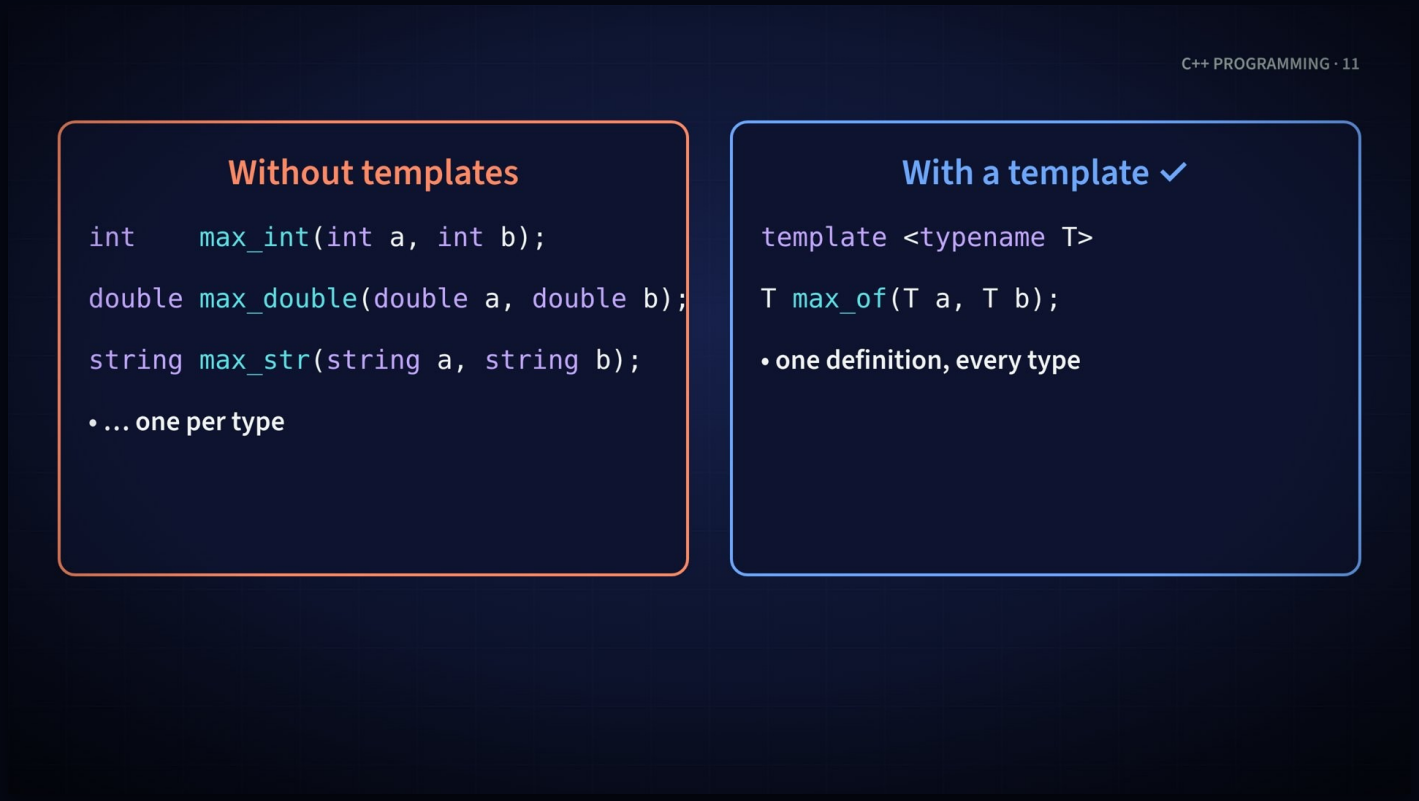
C++ PROGRAMMING

LESSON 11

Templates and Generic Programming

Write once, use with any type

The Problem



- In C, a function to find the larger of two values must be written once for int, again for …
- Templates solve this. You write the code once with a placeholder type, and the compiler …

11.01

Function Templates

max.cpp

```cpp
template <typename T>
T max_of(const T& a, const T& b) {
    return (a > b) ? a : b;
}

max_of(3, 7);                       // T = int     → 7
max_of(2.5, 1.5);                   // T = double  → 2.5
max_of(std::string{"Kabul"}, std::string{"Herat"});   // "Kabul"
```

- A function template starts with template and a type parameter in angle brackets, usually …
- When you call it, the compiler deduces T from the arguments and creates the right version. …

11.02

Class Templates

stack.hpp

```
template <typename T>
class Stack {
public:
    void push(const T& item) { items.push_back(item); }
    T pop() { T top = items.back(); items.pop_back(); return top; }
    bool empty() const { return items.empty(); }
private:
    std::vector<T> items;
};

Stack<int> numbers;     numbers.push(42);
Stack<std::string> words;   words.push("salaam");
```

- Classes can be templates too. A Stack of T works for ints, strings or students. You choose …
- This is exactly how std vector, std map and the whole standard library are built: vector of …

11.03

Concepts: Constraining Templates

concepts.cpp

```cpp
#include <concepts>

template <std::totally_ordered T>
T max_of(const T& a, const T& b) { return (a > b) ? a : b; }

template <typename T>
concept Shape = requires(const T& s) { { s.area() } -> std::convertible_to<double>; };

double total_area(const Shape auto& a, const Shape auto& b) {
    return a.area() + b.area();
}
```

- What if someone calls max of with a type that cannot be compared? Old compilers printed …
- C plus plus 20 added concepts, which state the requirements of a template. If a type does …

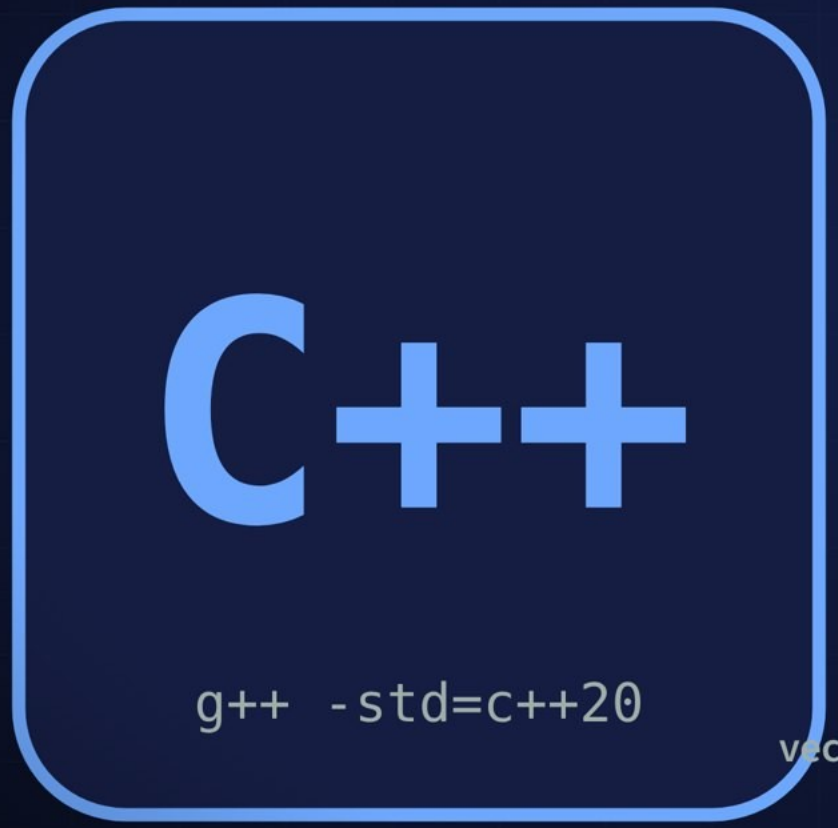
11.04

Templates Live in Headers



- In headers — full code visible
- Zero run-time cost — generated at compile time
- Trade-off — slower builds, bigger binaries

11.05



A FREE MYWEBUNIVERSITY COURSE

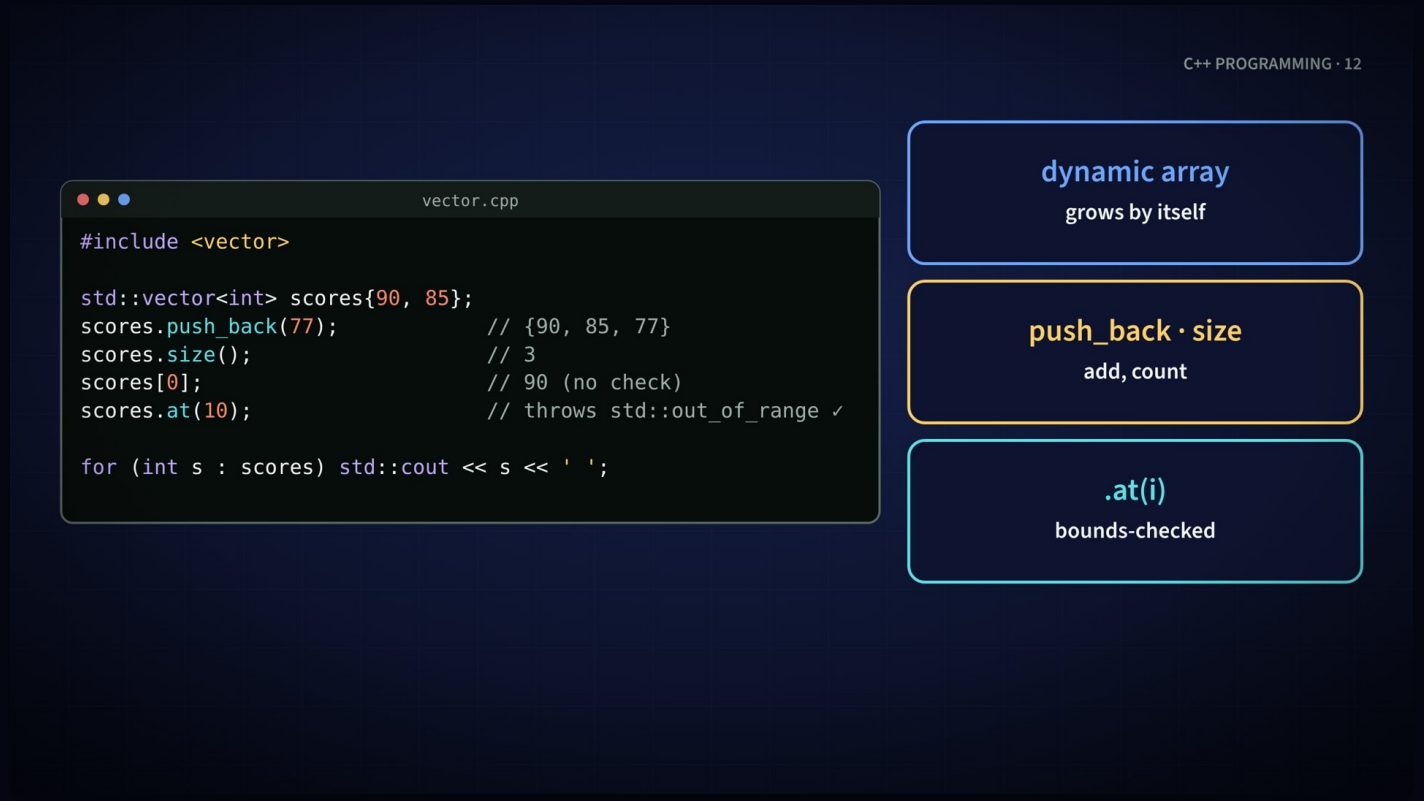
C++ PROGRAMMING

LESSON 12

STL Containers: vector, map and More

vector, array, multi-dimensional arrays, map, unordered_map, set, stack and queue

std::vector



- dynamic array — grows by itself
- push_back · size — add, count
- .at(i) — bounds-checked

12.01

std::array and Multi-Dimensional Arrays

grids.cpp

```cpp
#include <array>

std::array<int, 5> a{1, 2, 3, 4, 5};      // fixed size, a.size() == 5
std::array<std::array<int, 3>, 3> board{}; // 3×3, e.g. tic-tac-toe

int rows = 4, cols = 6;
std::vector<std::vector<double>> grid(rows, std::vector<double>(cols, 0.0));
grid[2][3] = 1.5;

std::vector<double> flat(rows * cols);      // fast: one block
flat[2 * cols + 3] = 1.5;
```

- std array is a fixed-size array that knows its own size and works with the standard …
- For a grid whose size is known only at run time, use a vector of vectors. For high …

12.02

std::map and std::unordered_map

map.cpp

```cpp
#include <map>
#include <unordered_map>

std::map<std::string, int> grades;
grades["Mariam"] = 95;
grades["Ahmad"]  = 88;
for (const auto& [name, g] : grades)        // sorted by name
    std::cout << name << ": " << g << "\n";

std::unordered_map<std::string, int> word_count;   // hash table
word_count["salaam"]++;
if (word_count.contains("kabul")) { }             // C++20
```

- A map stores key-value pairs, like a dictionary: a word and its meaning, or a student name …
- std map keeps keys sorted and finds them in logarithmic time. std unordered map uses a hash …
- Square brackets insert a missing key automatically. Use find or contains when you only want …

12.03

set, stack, queue and More

more.cpp

```cpp
std::set<int> ids{3, 1, 3, 2};       // {1, 2, 3}

std::stack<int> s;  s.push(1); s.push(2);
s.top();            // 2  (last in, first out)

std::queue<std::string> q;
q.push("first"); q.push("second");
q.front();          // "first"

std::priority_queue<int> pq;  pq.push(5); pq.push(9); pq.push(2);
pq.top();           // 9
```

- A set stores unique values, ignoring duplicates. Like map, it comes in sorted and unordered …
- stack gives last-in-first-out access, queue gives first-in-first-out, and priority queue …

12.04

Choosing a Container

Container	Best for
`vector`	the default: fast, contiguous
`array`	fixed size known at compile time
`map / unordered_map`	look up values by key
`set / unordered_set`	unique values, fast membership
`deque`	add/remove at both ends
`stack · queue · priority_queue`	LIFO · FIFO · largest first

- Start with vector; it is the fastest for most jobs, because its elements sit together in …
- Use deque when you add at both ends, and list only when you constantly insert in the middle …

12.05



A FREE MYWEBUNIVERSITY COURSE

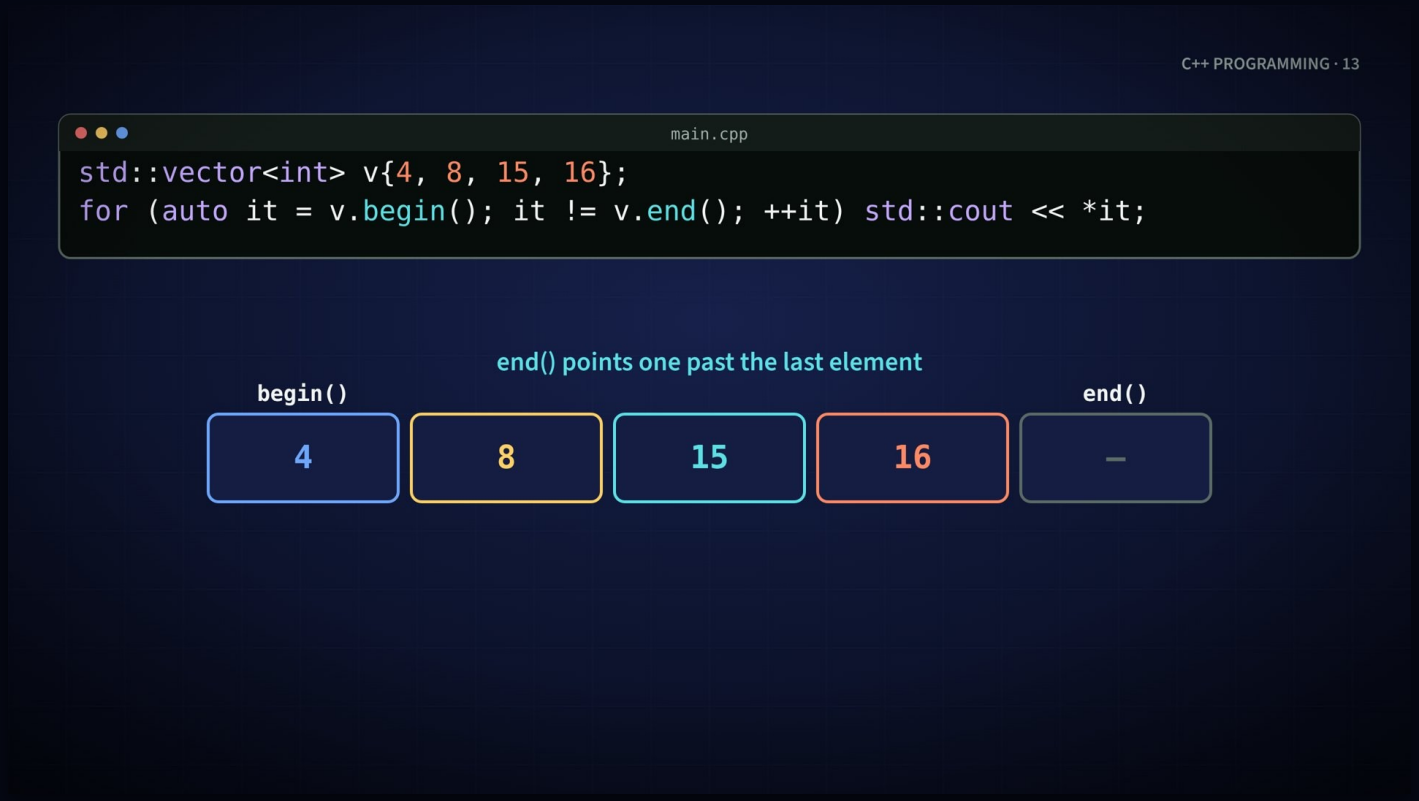
C++ PROGRAMMING

LESSON 13

Iterators and Algorithms

sort, find, count, accumulate, transform and ranges

Iterators



- An iterator is like a smart pointer into a container. begin points to the first element, …
- Every container provides iterators, so the same algorithm works on a vector, a list or a …

13.01

sort and find

algorithms.cpp

```cpp
#include <algorithm>

std::vector<int> v{42, 7, 19, 3};
std::sort(v.begin(), v.end());                    // 3 7 19 42
std::sort(v.begin(), v.end(), std::greater<>());   // 42 19 7 3

auto it = std::find(v.begin(), v.end(), 19);
if (it != v.end()) std::cout << "found";

auto big = std::find_if(v.begin(), v.end(), [](int x) { return x > 10; });
```

- Include algorithm, and you get dozens of tested, optimized algorithms. sort orders a range; …
- find searches for a value and returns an iterator to it, or end if it is missing. find if …

13.02

count, accumulate and transform

numeric.cpp

```cpp
#include <numeric>

std::vector<int> scores{90, 55, 77, 64, 98};
auto passed = std::count_if(scores.begin(), scores.end(),
                            [](int s) { return s >= 60; });     // 4
int total = std::accumulate(scores.begin(), scores.end(), 0);   // 384

std::vector<int> curved(scores.size());
std::transform(scores.begin(), scores.end(), curved.begin(),
               [](int s) { return std::min(s + 5, 100); });
```

- count if counts the elements that match a condition. accumulate, from numeric, adds …
- transform applies a function to every element and writes the results. These one-line calls …

13.03

C++20 Ranges

ranges.cpp

```cpp
#include <ranges>

std::ranges::sort(scores);                  // no begin/end

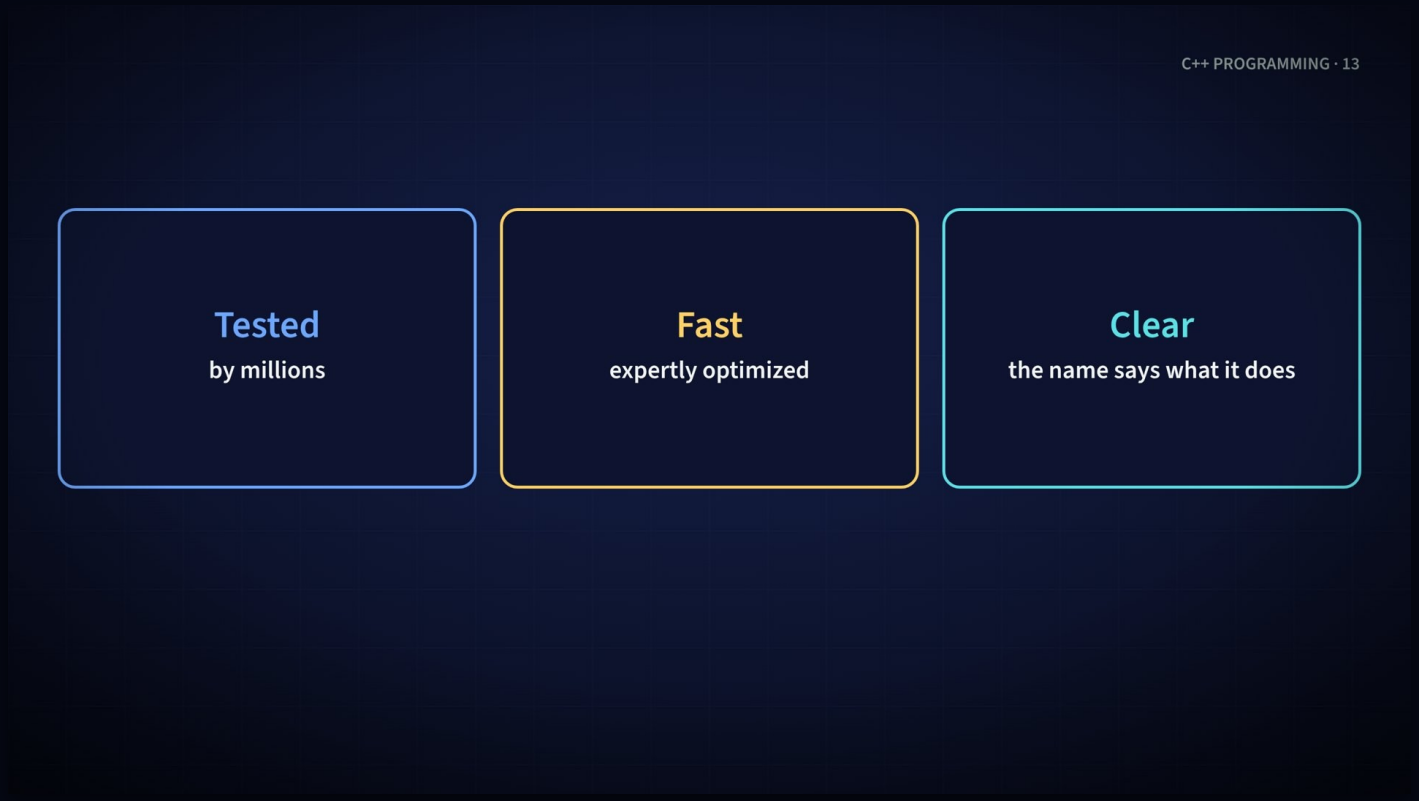
auto result = scores
    | std::views::filter([](int s) { return s >= 60; })
    | std::views::transform([](int s) { return s + 5; });

for (int s : result) std::cout << s << ' ';
```

- C plus plus 20 ranges let you pass the whole container instead of begin and end. They also …
- The result reads like a sentence: take the scores, keep the passing ones, add five, and …

13.04

Why Use Algorithms?



- Tested — by millions
- Fast — expertly optimized
- Clear — the name says what it does

13.05



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 14

Strings and File Streams

std::string methods, fstream, stringstream and formatting

More std::string

strings.cpp

```cpp
std::string email = "wahid@mywebuniversity.com";
auto at = email.find('@');                    // 5
std::string user = email.substr(0, at);       // "wahid"
bool edu = email.ends_with(".com");           // true (C++20)

int n = std::stoi("2026");                    // text → int
std::string s = std::to_string(3.5);          // "3.500000"

void greet(std::string_view name);            // no copy
```

- std string has many useful methods: substr cuts a part, find searches, replace changes …
- Convert numbers to text with to string, and text to numbers with stoi and stod. std string …

14.01

Writing Files with ofstream

write.cpp

```cpp
#include <fstream>

{
    std::ofstream out{"scores.txt"};
    if (!out) {
        std::cerr << "cannot open file\n";
        return 1;
    }
    out << "Mariam " << 95 << "\n";
    out << "Ahmad "  << 88 << "\n";
}                       // closed automatically (RAII)
```

- Include fstream. An ofstream object opens a file for writing, and you write to it with the …
- Thanks to RAII, the file closes automatically when the object goes out of scope. Check that …

14.02

Reading Files with ifstream

read.cpp

```cpp
std::ifstream in{"scores.txt"};
std::string name;
int score;
while (in >> name >> score) {            // word, then number
    std::cout << name << ": " << score << "\n";
}

std::ifstream text{"notes.txt"};
std::string line;
while (std::getline(text, line)) { /* one line at a time */ }
```

- An ifstream reads a file. Use getline in a loop to read one line at a time, or the …
- The loop condition is the read itself, so it stops cleanly at the end of the file.

14.03

String Streams

csv.cpp

```cpp
#include <sstream>

std::istringstream row{"Herat,2026,95.5"};
std::string city, year, score;
std::getline(row, city, ',');
std::getline(row, year, ',');
std::getline(row, score, ',');

std::ostringstream msg;
msg << city << " scored " << score;
std::string text = msg.str();    // "Herat scored 95.5"
```

- A stringstream treats a string like a stream. Use istringstream to split a line into parts, …
- They are very handy for parsing CSV lines and for building messages.

14.04

Formatting Output

format.cpp

```cpp
#include <iomanip>
#include <format>

std::cout << std::setw(10) << "Mariam" << std::fixed << std::setprecision(2) << 95.456;
//     Mariam95.46

std::string s = std::format("{:<10}{:>6.2f}", "Ahmad", 88.5);   // C++20
std::println("Total: {}", 384);                                 // C++23
```

- iomanip gives stream manipulators like setw for column width and setprecision for decimal …
- C plus plus 20 added std format, which uses curly-brace placeholders like Python, and C …

14.05



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 15

Exceptions and Error Handling

try, catch, throw, standard exceptions, noexcept and RAII safety

The C Way: Return Codes

c_errors.c

```
FILE *f = fopen("data.txt", "r");
if (f == NULL) { return -1; }          // check
char *buf = malloc(1000);
if (buf == NULL) { fclose(f); return -1; }   // check + clean up
if (fread(buf, 1, 1000, f) == 0) {
    free(buf); fclose(f); return -1;           // check + clean up again
}
/* ... the real work is buried in checks ... */
```

- In C, a function reports an error by returning a special value, like minus one or a null …
- Programmers forget to check, and the error silently travels on and causes a crash far away. …

15.01

try, throw and catch

exceptions.cpp

```cpp
double divide(double a, double b) {
    if (b == 0)
        throw std::invalid_argument("division by zero");
    return a / b;
}

try {
    std::cout << divide(10, 0);
} catch (const std::invalid_argument& e) {
    std::cerr << "Error: " << e.what() << "\n";
}
```

- C plus plus separates the error path from the normal path. When something goes wrong, the …
- The exception jumps up the call stack until a matching catch block handles it. The code in …

15.02

Standard Exception Types

Exception	Thrown when
`std::out_of_range`	vector.at(i) with a bad index
`std::invalid_argument`	stoi("abc") — bad input
`std::bad_alloc`	new cannot get memory
`std::runtime_error`	general errors at run time
`std::exception`	base class: catches them all

- The standard library has a family of exception classes, all inheriting from std exception, …
- Catch the most specific types first, and always catch by const reference to avoid copying …

15.03

Your Own Exceptions

bank.cpp

```cpp
class InsufficientFunds : public std::runtime_error {
public:
    InsufficientFunds()
        : std::runtime_error("not enough money") {}
};

void withdraw(double amount) {
    if (amount > balance) throw InsufficientFunds{};
    balance -= amount;
}

double get_balance() const noexcept { return balance; }
```

- You can create your own exception class by inheriting from std runtime error. This lets …
- Mark functions that never throw with noexcept. It is a promise to the compiler and to …

15.04

Exceptions + RAII = Safe Code

safe.cpp

```cpp
void process() {
    std::ifstream f{"data.txt"};          // closes itself
    std::vector<char> buf(1000);          // frees itself
    auto p = std::make_unique<Parser>();  // deletes itself

    if (!f) throw std::runtime_error("cannot open");
    // ... real work, no clean-up code ...
}   // error or not: everything is released
```

- When an exception leaves a function, C plus plus destroys every local object on the way …
- So if your resources live in RAII objects, like vector, string, fstream and unique pointer, …

15.05



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 16

Data Structures in C++

Linked list class, stack, queue, binary search tree, hash table and Big-O

A Linked List Class

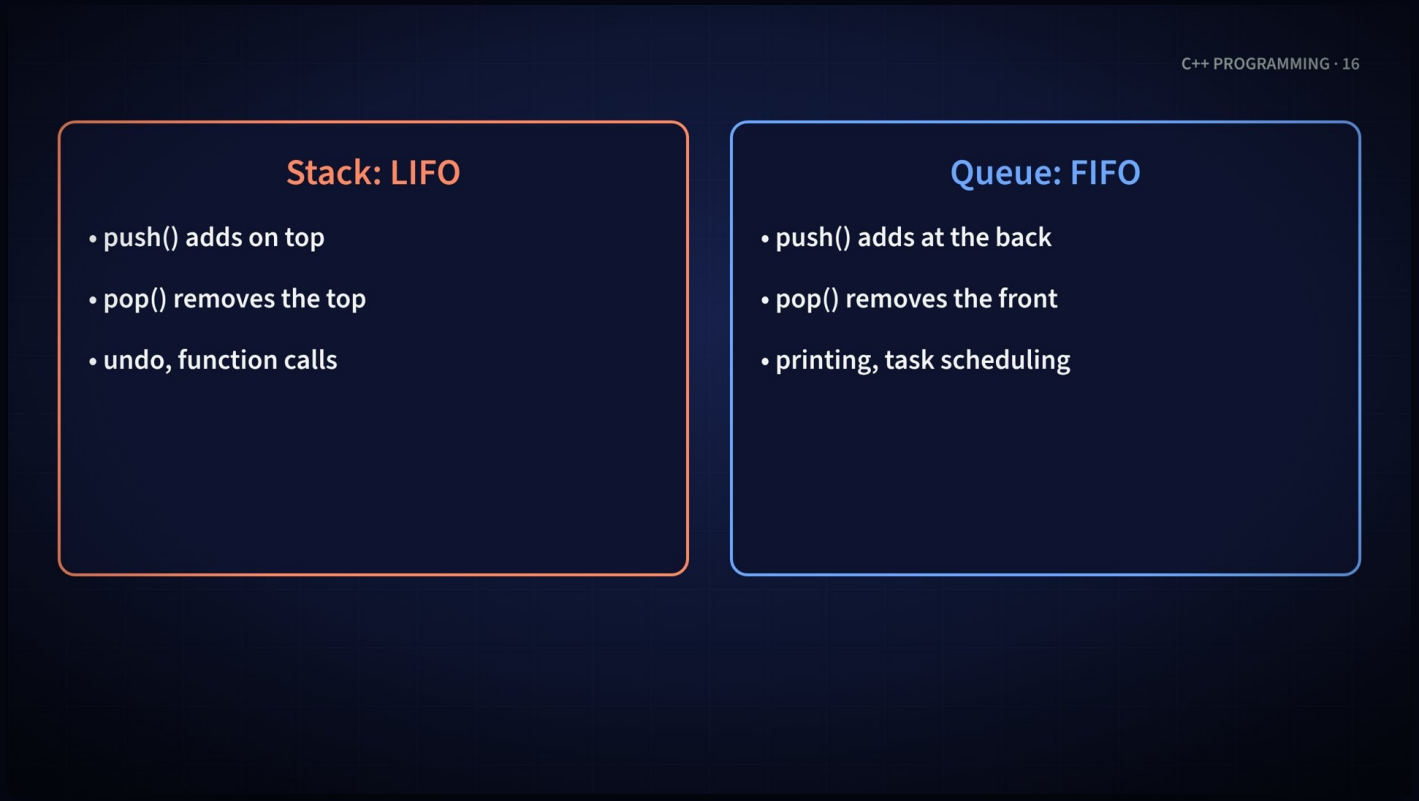
list.cpp

```cpp
template <typename T>
class LinkedList {
    struct Node {
        T data;
        std::unique_ptr<Node> next;
    };
    std::unique_ptr<Node> head;
public:
    void push_front(T value) {
        head = std::make_unique<Node>(
            Node{value, std::move(head)});
    }
};
```

- In C, our linked list was a struct and loose functions. In C plus plus, it becomes one …
- With unique pointer for next, each node owns the one after it, so the whole chain frees …

16.01

Stack and Queue



- A stack is last in, first out, like a pile of plates: push on top, pop from the top. It ...
- A queue is first in, first out, like a line at a bank: push at the back, pop from the ...

16.02

Binary Search Tree

bst.cpp

```cpp
struct TreeNode {
    int value;
    std::unique_ptr<TreeNode> left, right;
};

bool contains(const TreeNode* n, int x) {
    while (n) {
        if (x == n->value) return true;
        n = (x < n->value) ? n->left.get()     // go left
                           : n->right.get();   // go right
    }
    return false;
}
```

- In a binary search tree, each node has up to two children. Smaller values go left, larger …
- To find a value, you go left or right at each step, so you skip half the tree each time. A …

16.03

Hash Tables

main.cpp

```
std::unordered_map<std::string, int> phone;
phone["Mariam"] = 5551;   // hash("Mariam") → slot 2
int n = phone["Mariam"];  // jump to slot 2
```



- A hash table turns a key into an array index using a hash function, so it jumps straight to …
- Lookup is about constant time on average, the fastest of all. In C plus plus, std unordered …

16.04

Choosing with Big-O

Structure	Find	Insert	Access [i]
vector	O(n)	O(1) at end	O(1)
list	O(n)	O(1)	O(n)
map / set	O(log n)	O(log n)	–
unordered_map	O(1)	O(1)	–
stack / queue	–	O(1)	top only

- Big O notation describes how the work grows as the data grows. Constant time stays flat, …
- This table is your guide: pick the structure whose fast operations match what your program …

16.05



A FREE MYWEBUNIVERSITY COURSE

C++ PROGRAMMING

LESSON 17

Modern C++, Projects and Next Steps

Move semantics, multi-file projects, CMake, compiler flags, sanitizers and careers

Move Semantics

move.cpp

```cpp
std::vector<int> big(1'000'000, 7);

std::vector<int> a = big;              // copy: 1 million elements
std::vector<int> b = std::move(big);   // move: just hand over the pointer
                                       // big is now empty

std::vector<int> make_data() {
    std::vector<int> v(1'000'000);
    return v;                          // moved or elided, never copied
}
```

- Copying a big vector copies every element, which is slow. Since C plus plus 11, an object …
- std move says: I am done with this, take its resources. Functions can also return big …

17.01

Modern C++ Toolbox

modern.cpp

```cpp
std::map<std::string, int> ages{{"Mariam", 21}, {"Ahmad", 25}};
for (const auto& [name, age] : ages)            // structured binding
    std::cout << name << " " << age << "\n";

std::optional<int> find_age(const std::string& n) {
    auto it = ages.find(n);
    if (it == ages.end()) return std::nullopt;  // maybe no value
    return it->second;
}
constexpr int square(int x) { return x * x; } // compile time
```

- Modern C plus plus gives you auto, range for loops, lambdas, smart pointers, constexpr for …
- C plus plus 20 adds concepts, ranges and modules. Write code with these, and your programs …

17.02

Multi-File Projects and CMake



- account.h — class declaration
- account.cpp — member functions
- main.cpp — uses the class

17.03

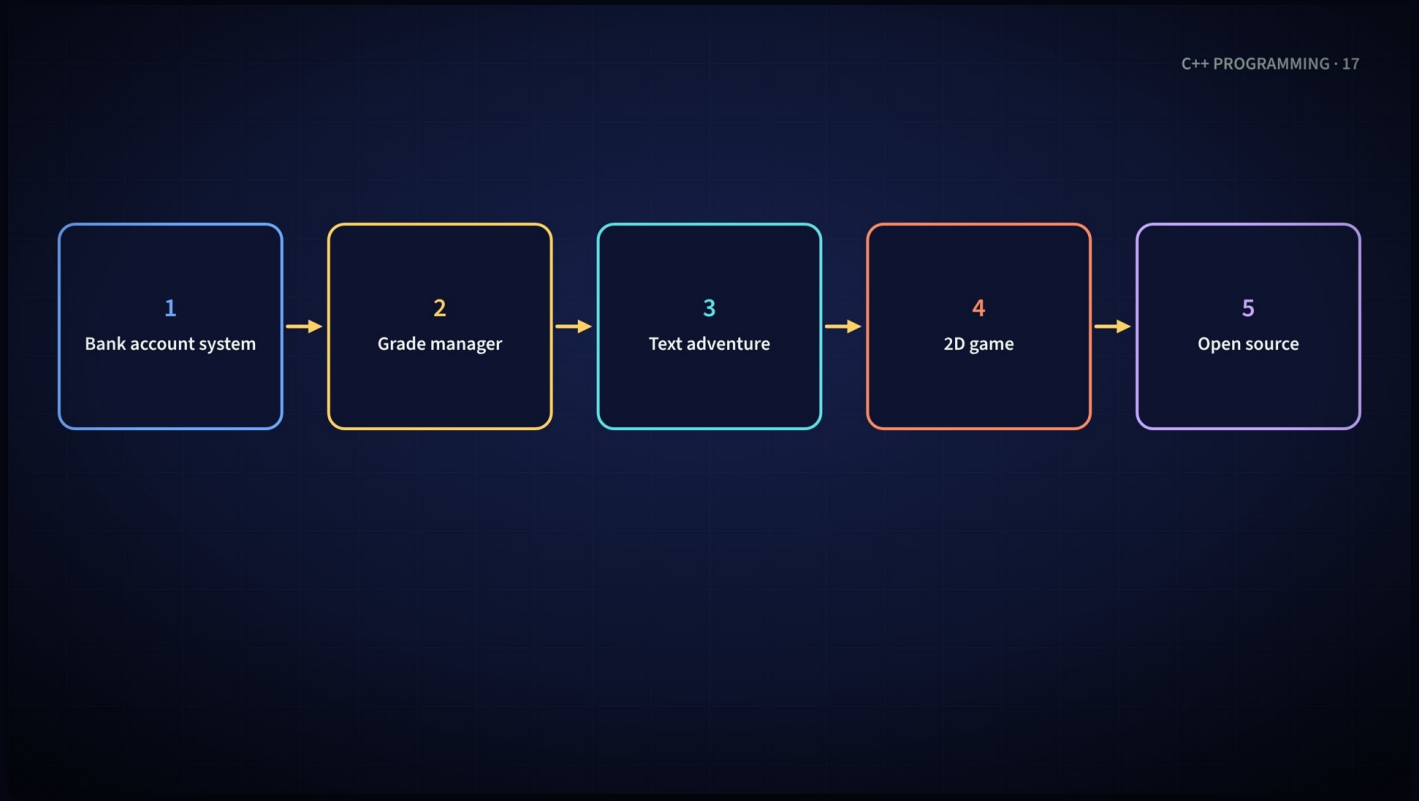
Compiler Flags and Sanitizers



- -Wall -Wextra — warnings
- -g — debug info
- -fsanitize — catch bugs

17.04

Projects and Careers



- 1 — Bank account system
- 2 — Grade manager
- 3 — Text adventure

17.05

Sources and Further Reading (1/1)

1. Bjarne Stroustrup, The C++ Programming Language, 4th ed., and A Tour of C++, 3rd ed.
2. ISO/IEC 14882 — the C++ standard (C++20, C++23)
3. cppreference.com — C++ language and library reference
4. C++ Core Guidelines — isocpp.github.io/CppCoreGuidelines
5. MyWebUniversity — free C++ and Linux courses

Thank you for learning with us

MyWebUniversity.com

Learn it all @MyWebUniversity

Free technical education since 2000 — English and Dari