Modern C++ Course

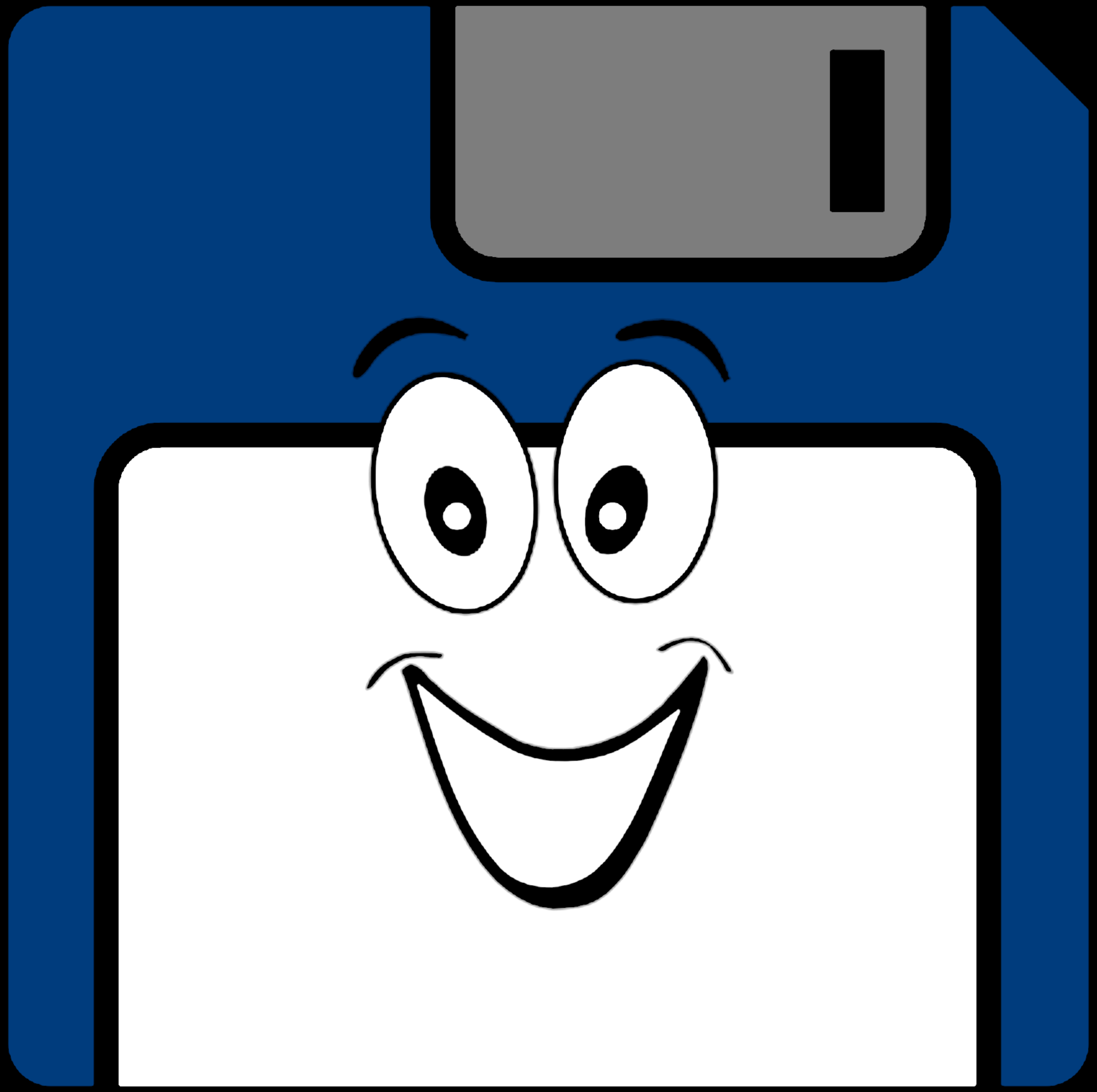
Who am I ?

Gammasoft

Gammasoft aims to make c++ fun again.

About

- Gammasoft is the nickname of Yves Fiumefreddo.
- More than thirty years of passion for high technology especially in development (c++, c#, objective-c, ...).
- Object-oriented programming is more than a mindset.
- more info see my GitHub : https://github.com/gammasoft71



Outline

Outline

Outline

Outline

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Classes (or “user-defined types”)

- structs on steroids
  - with inheritance
  - with access control
  - including methods (aka. member functions)

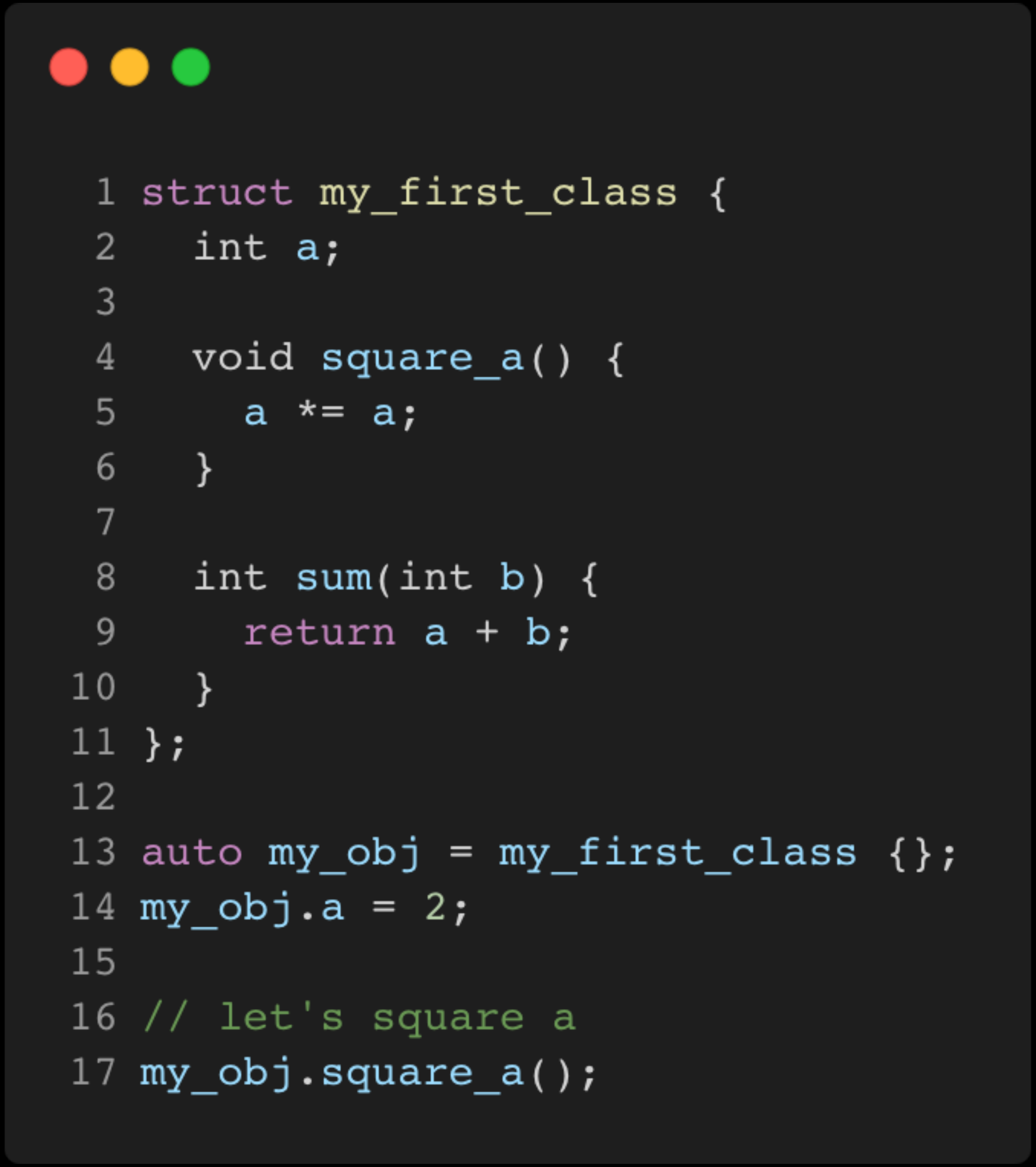
Objects

- instances of classes

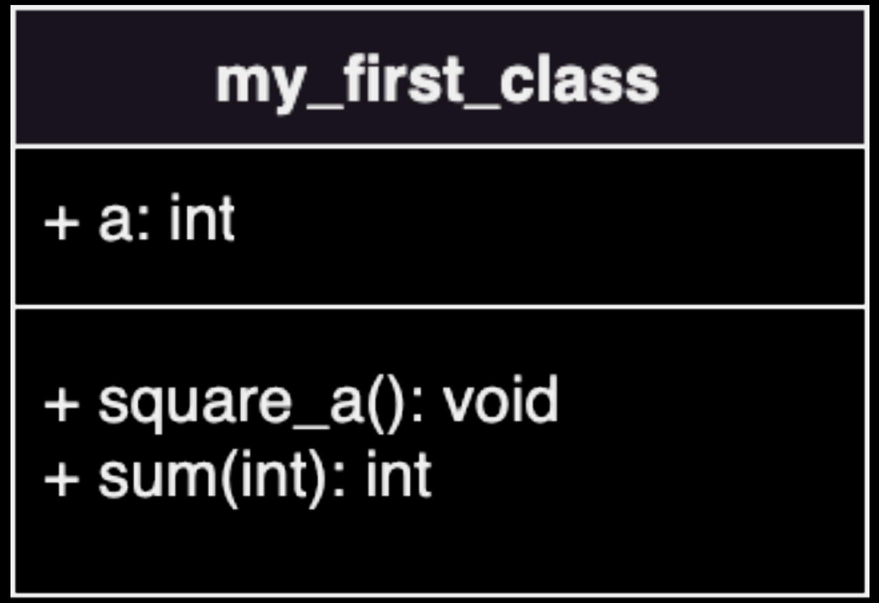
A class encapsulates state and behavior of “something”

- shows an interface
- provides its implementation
    - status, properties
    - possible interactions
    - construction and destruction

My first class



```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }

  int sum(int b) {
    return a + b;
  }
};

auto my_obj = my_first_class {};
my_obj.a = 2;

// let's square a
my_obj.square_a();
```



Separating the interface

Header: my_class.hpp

```cpp
#pragma once

struct my_class {
  int a;

  void square_a();
};
```

User 1: main.cpp

```cpp
#include <iostream>

int main() {
  auto mc = my_class {};
  //...
}
```

Implementation: my_class.cpp

```cpp
#include "my_class.hpp"

void my_class::square_a() {
  a *= a;
}
```

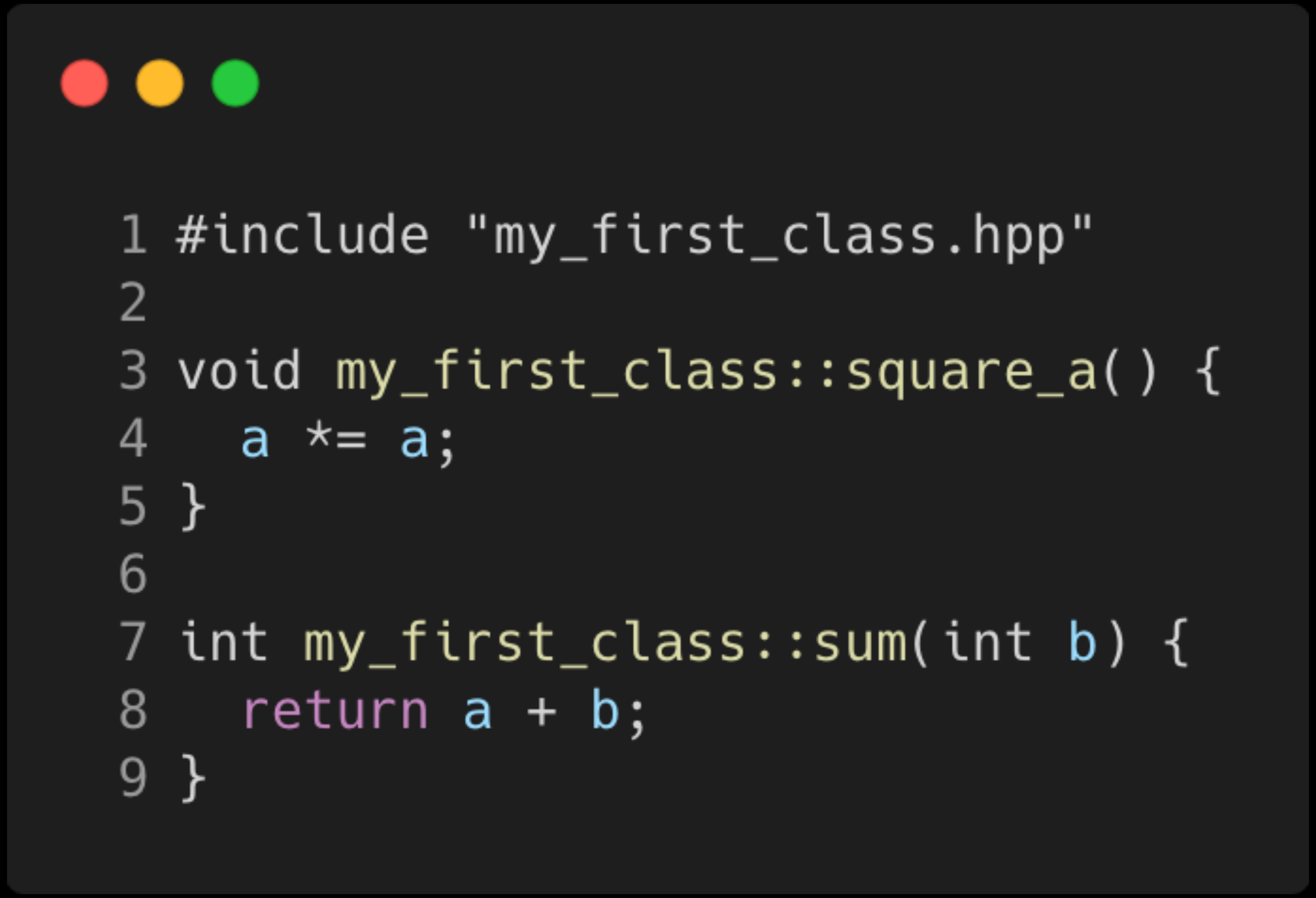
User 2: fun.cpp

```cpp
#include "my_class.hpp"

void fun(my_class& mc) {
  mc.square_a();
}
```

Implementing methods

Good practice

- usually in .cpp, outside of class declaration
- using the class name as "namespace"
- short member functions can be in the header
- some functions (templates, constexpr) must be in the header

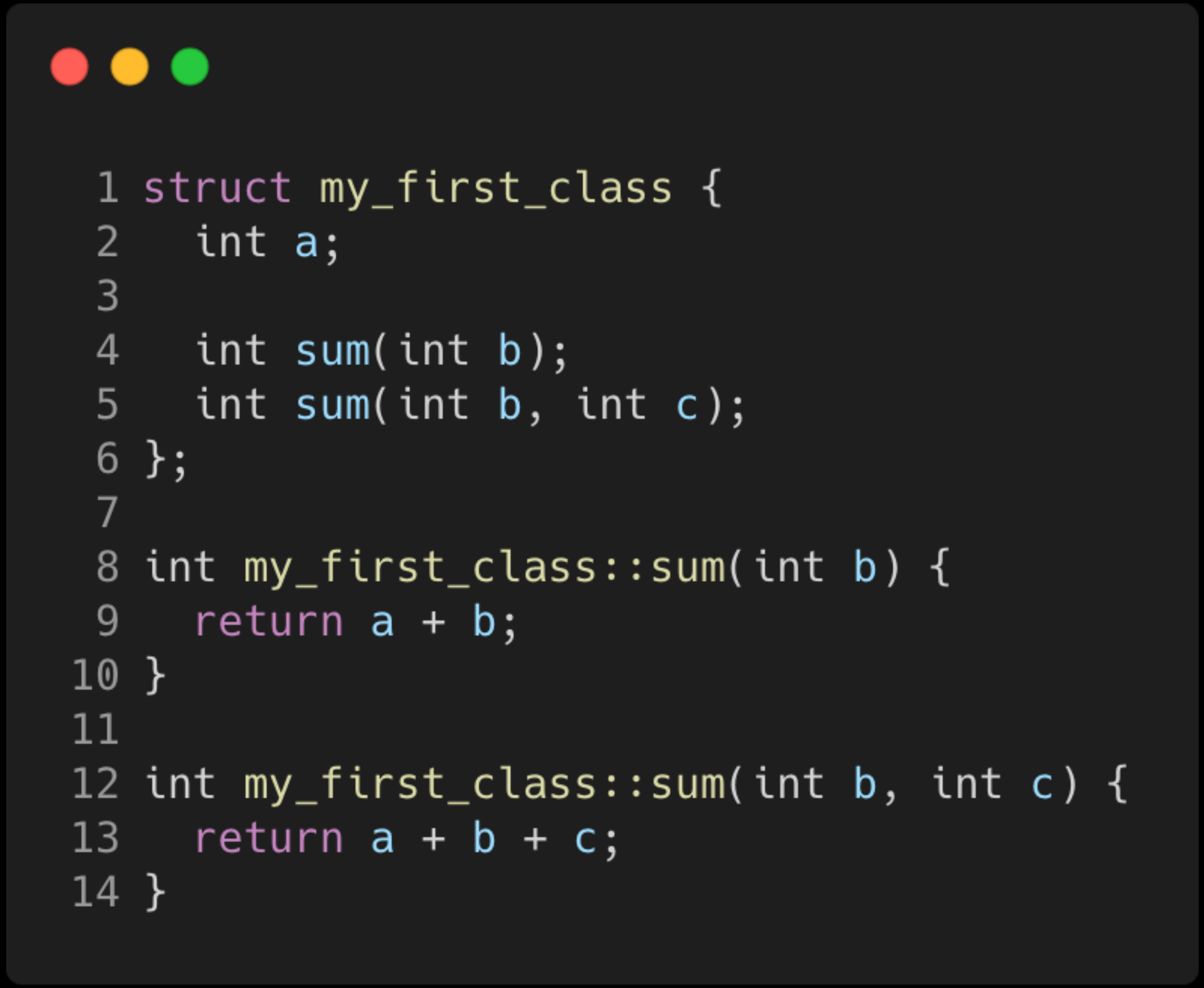


```cpp
#include "my_first_class.hpp"

void my_first_class::square_a() {
  a *= a;
}

int my_first_class::sum(int b) {
  return a + b;
}
```

Method overloading

The rules in C++

- overloading is authorized and welcome
- signature is part of the method identity
- but not the return type



```cpp
struct my_first_class {
  int a;

  int sum(int b);
  int sum(int b, int c);
};

int my_first_class::sum(int b) {
  return a + b;
}

int my_first_class::sum(int b, int c) {
  return a + b + c;
}
```

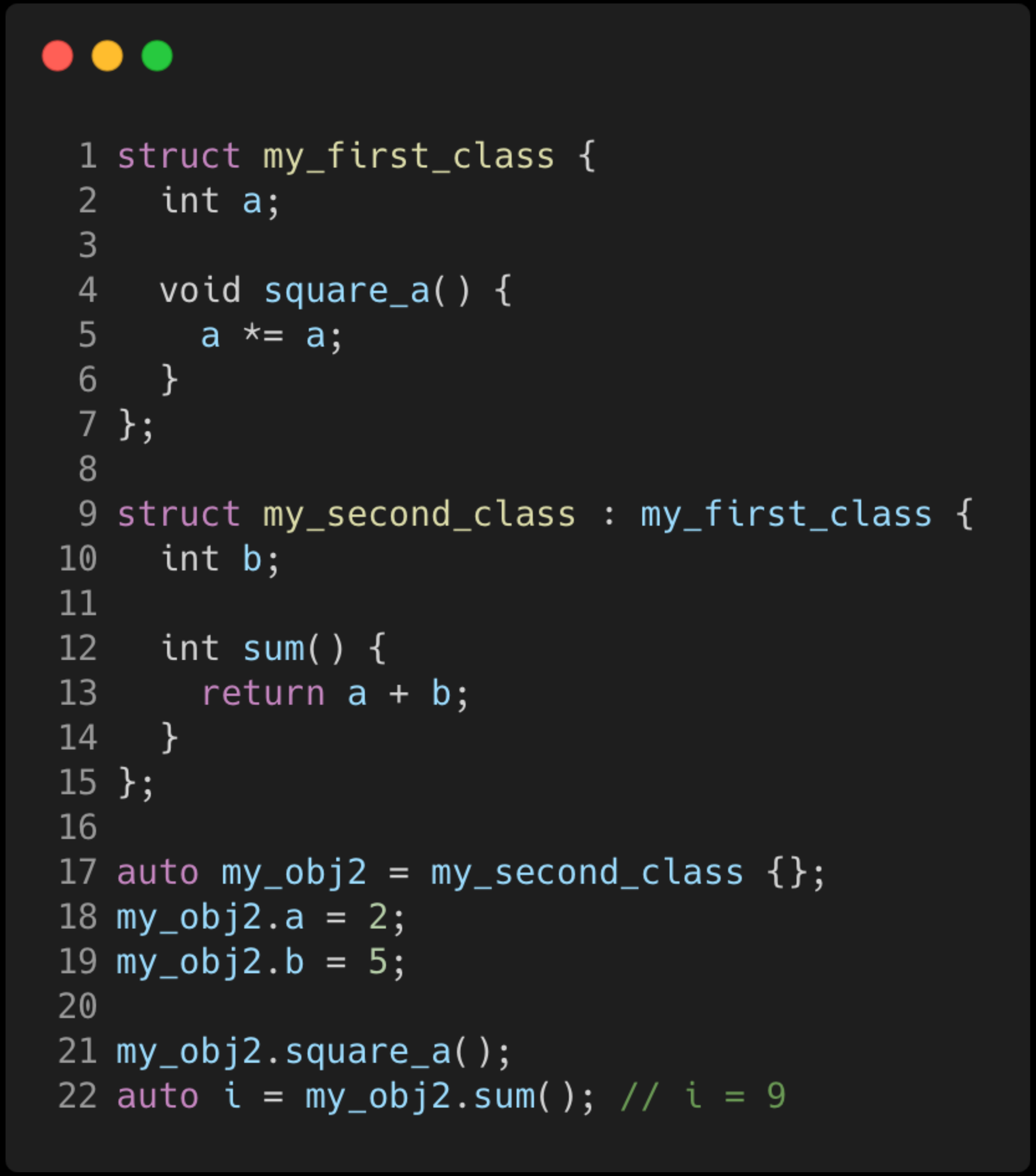
Objects Oriented Programming (OOP)

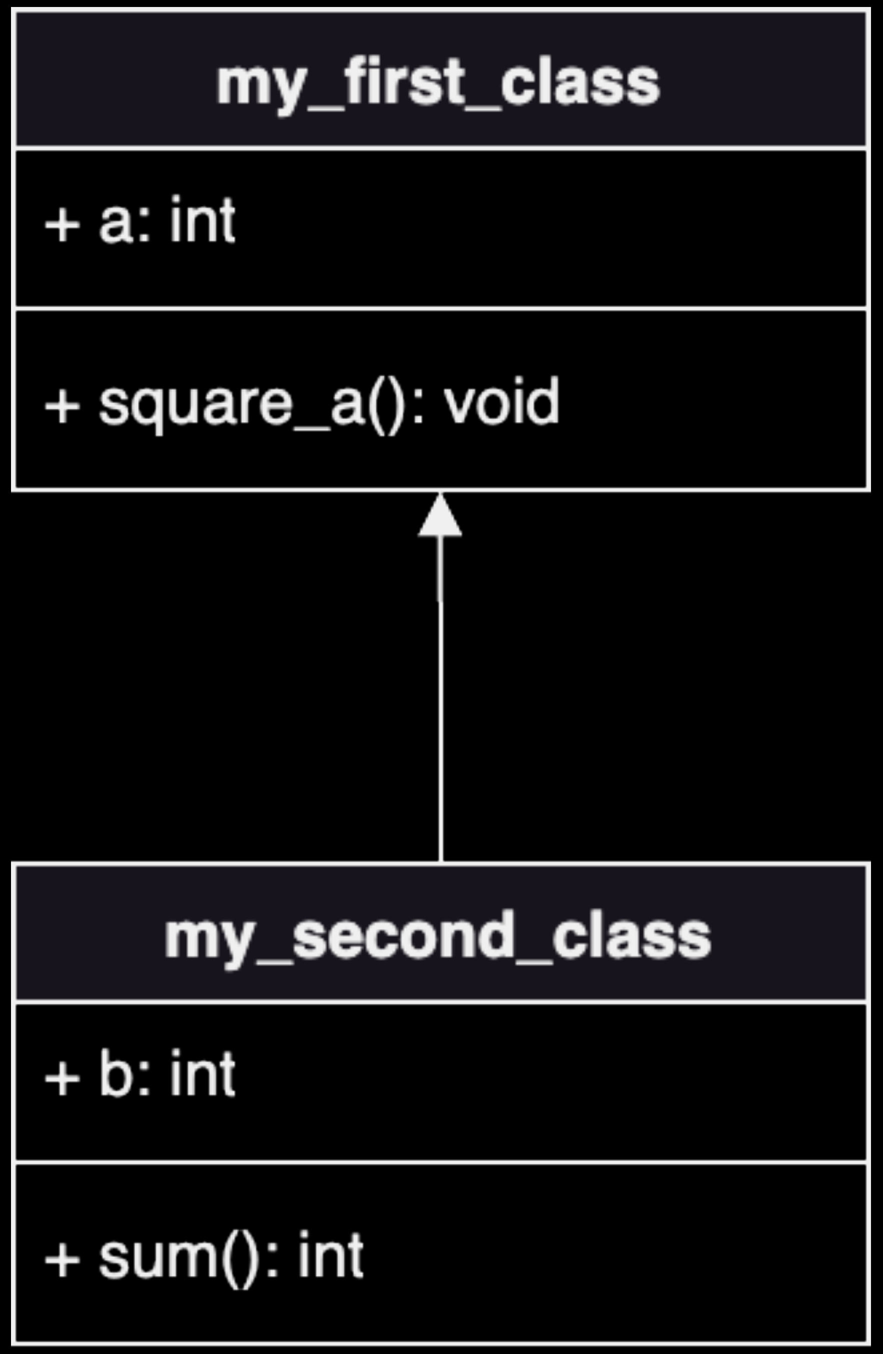
- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

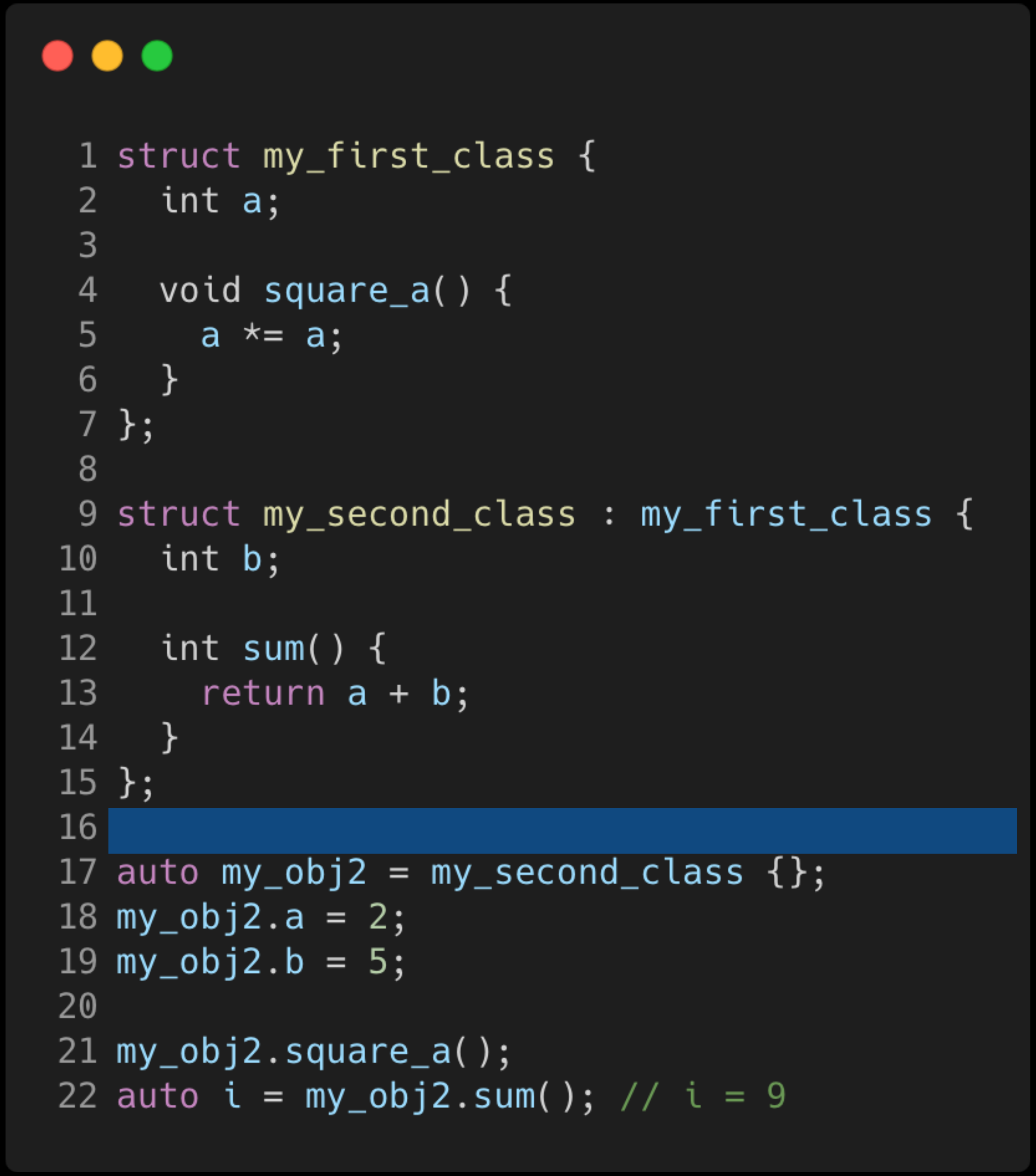
- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

First inheritance


```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

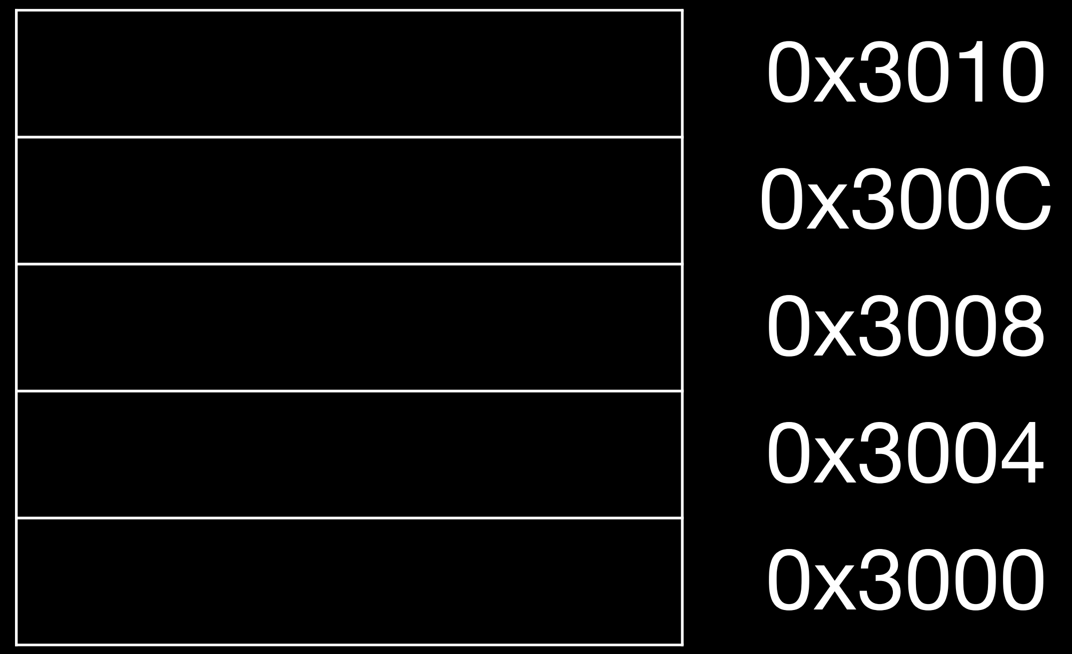


First inheritance

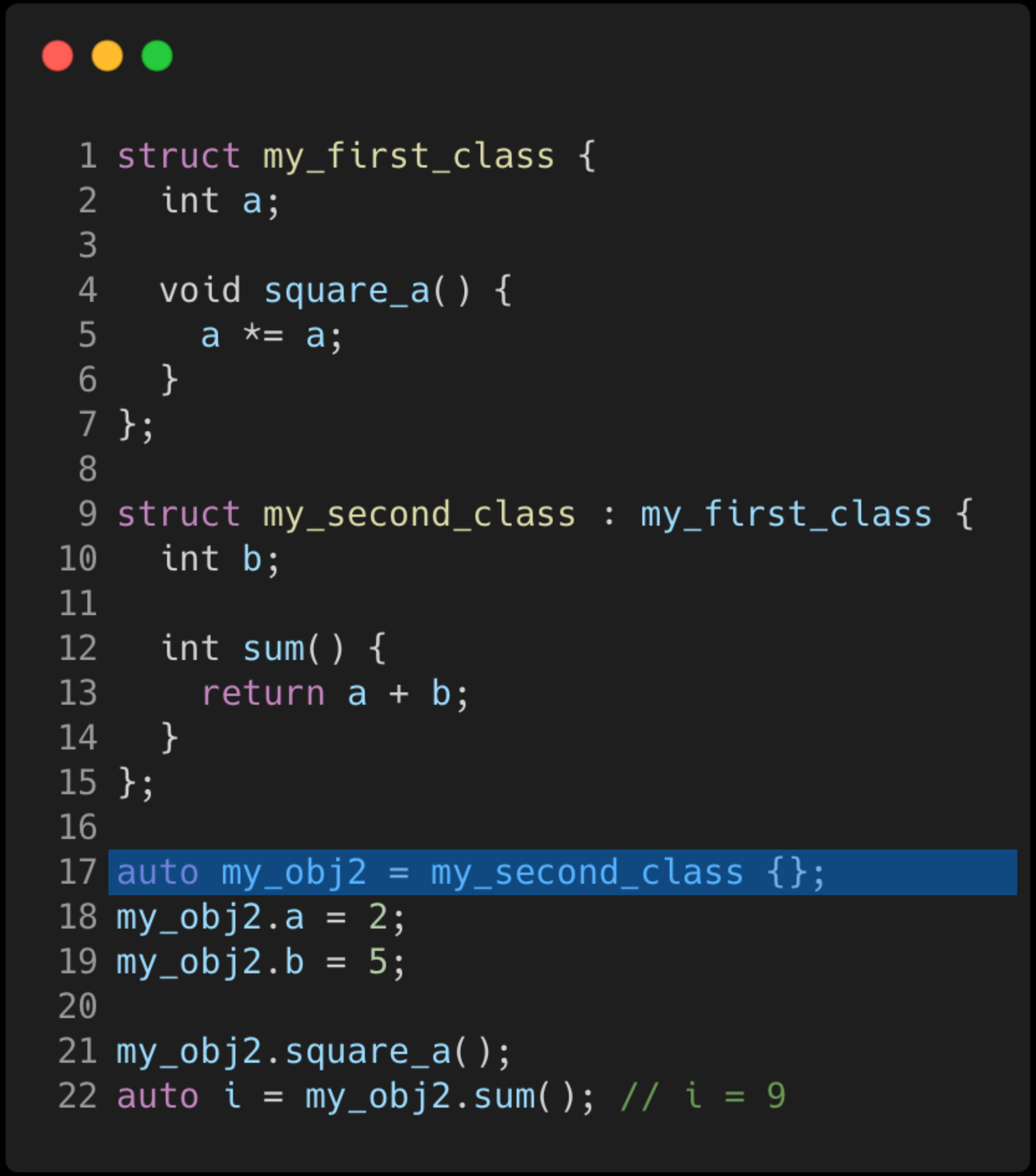


```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

Memory layout

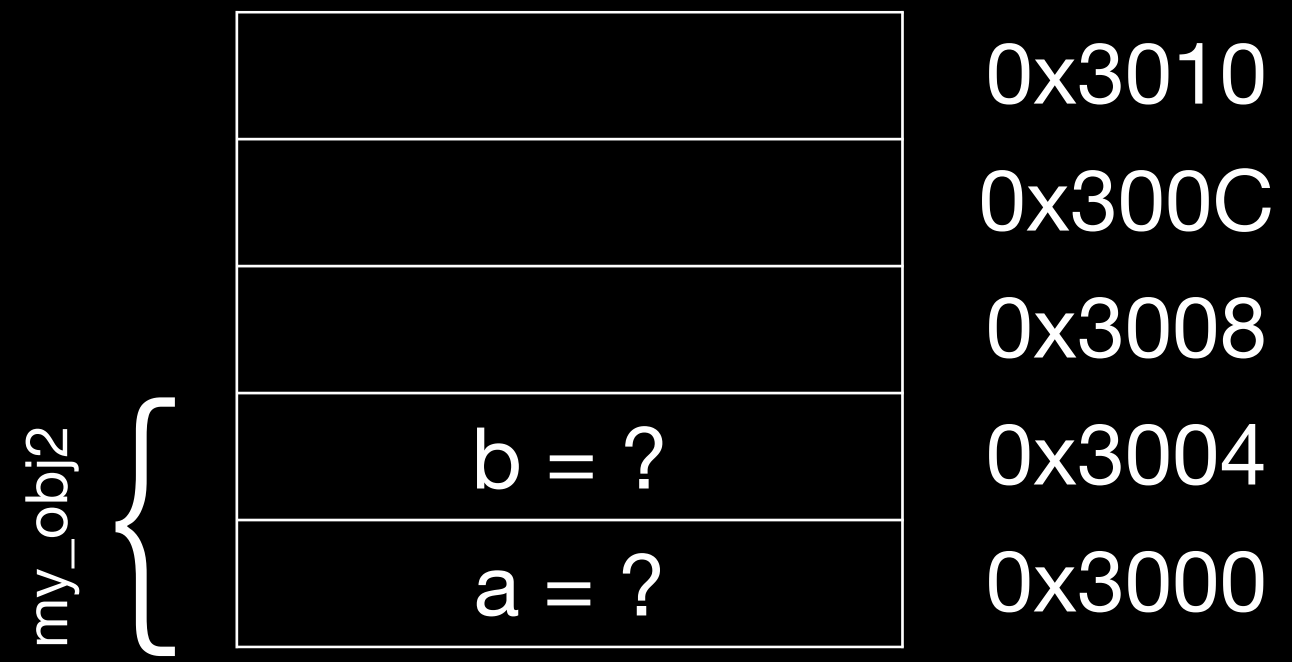


First inheritance

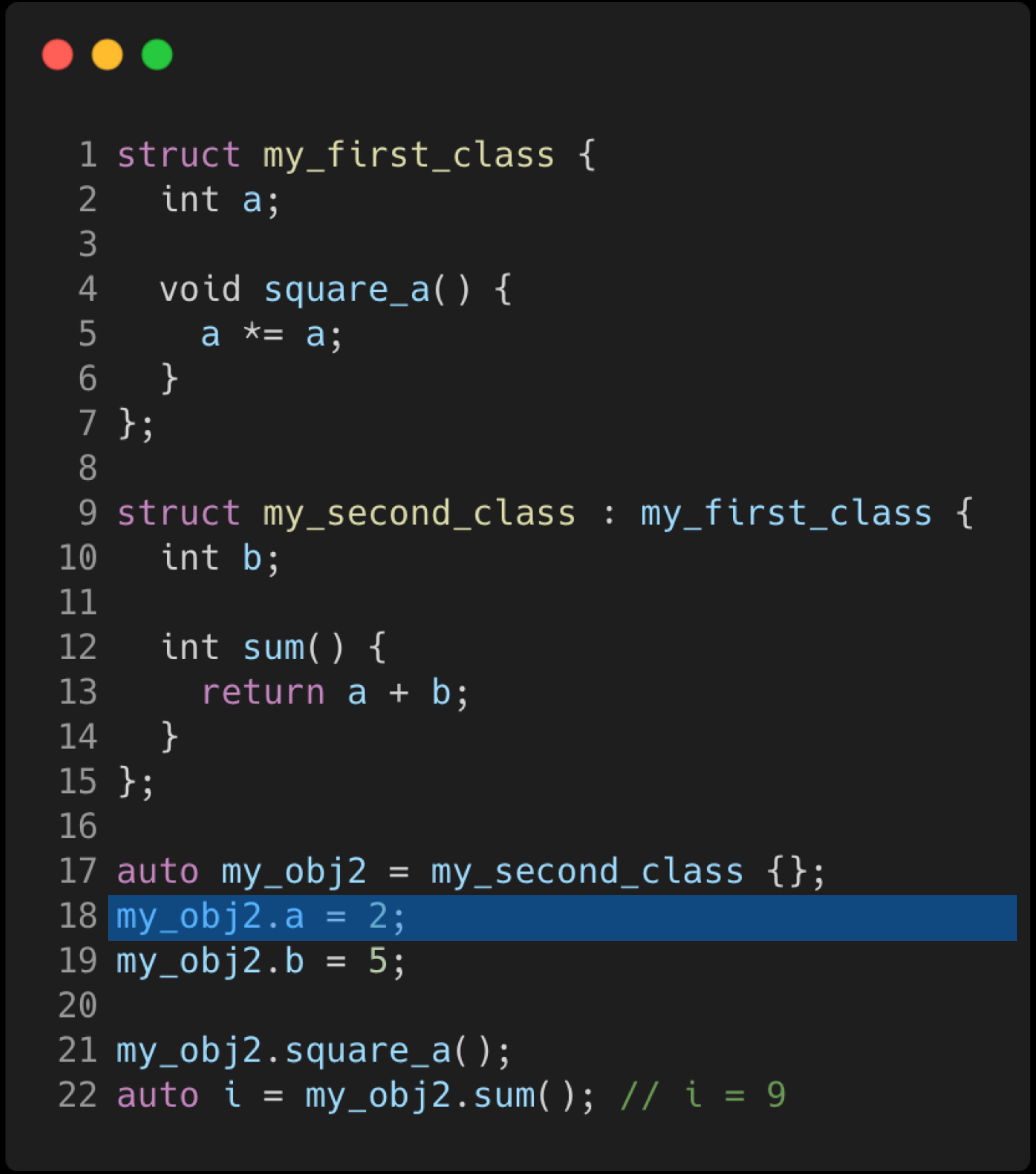


```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

Memory layout

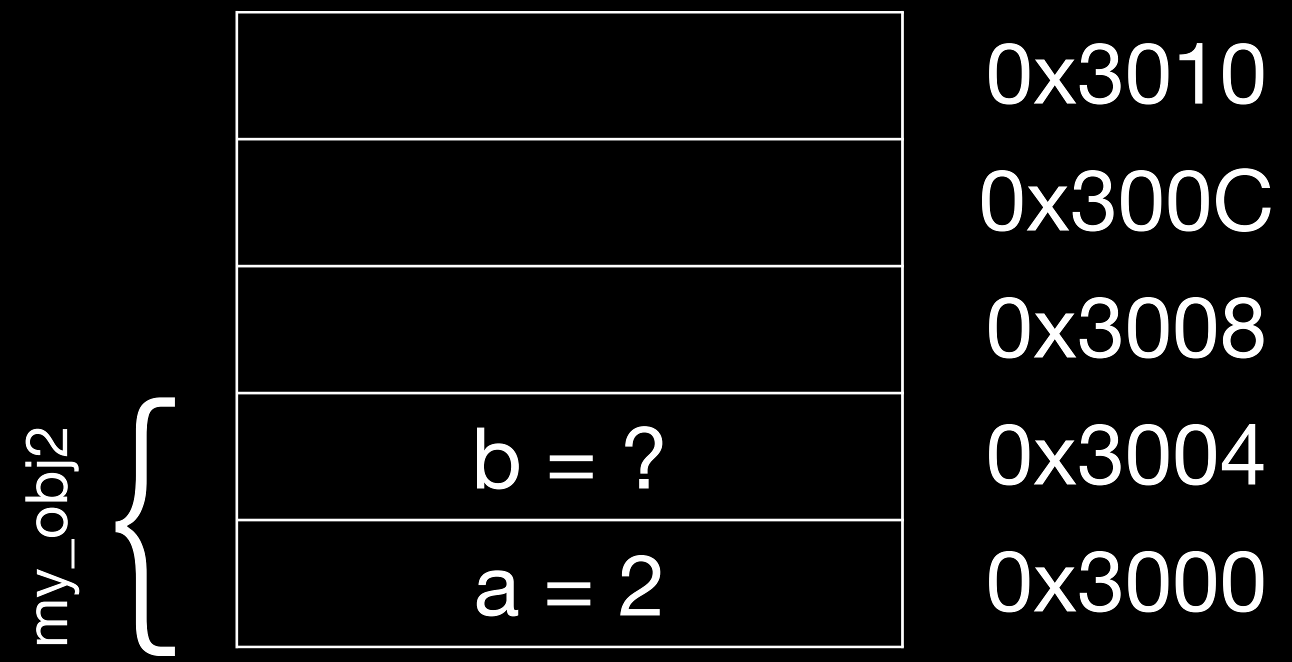


First inheritance



```
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

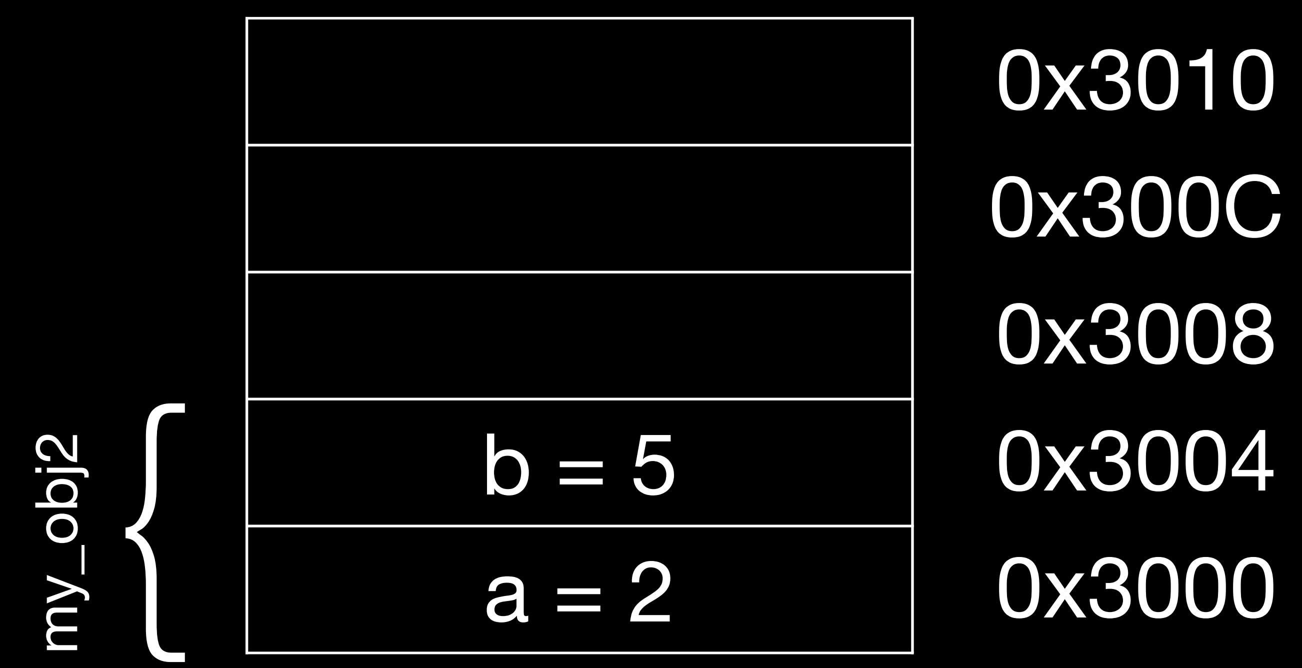
Memory layout



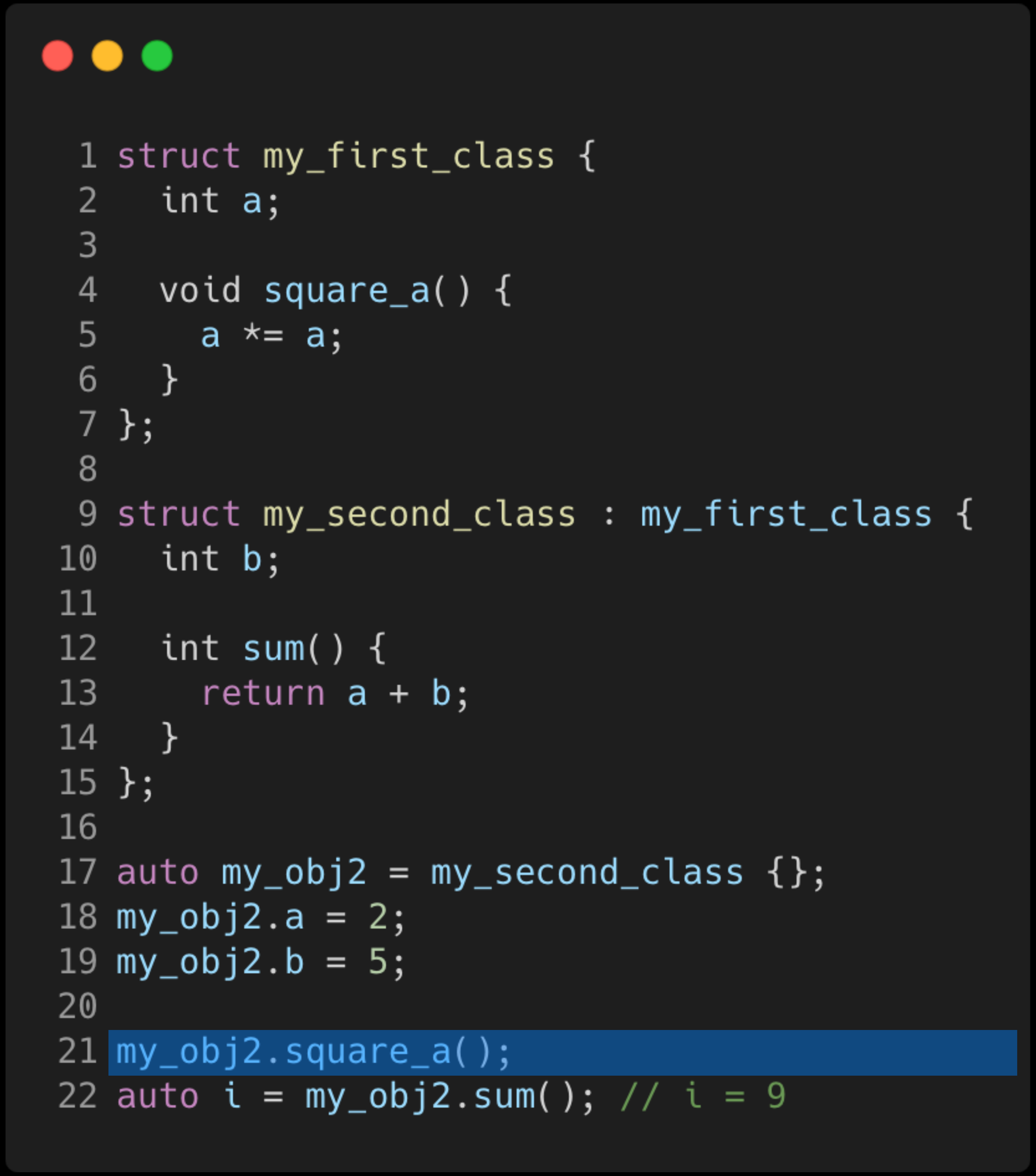
First inheritance

```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

Memory layout

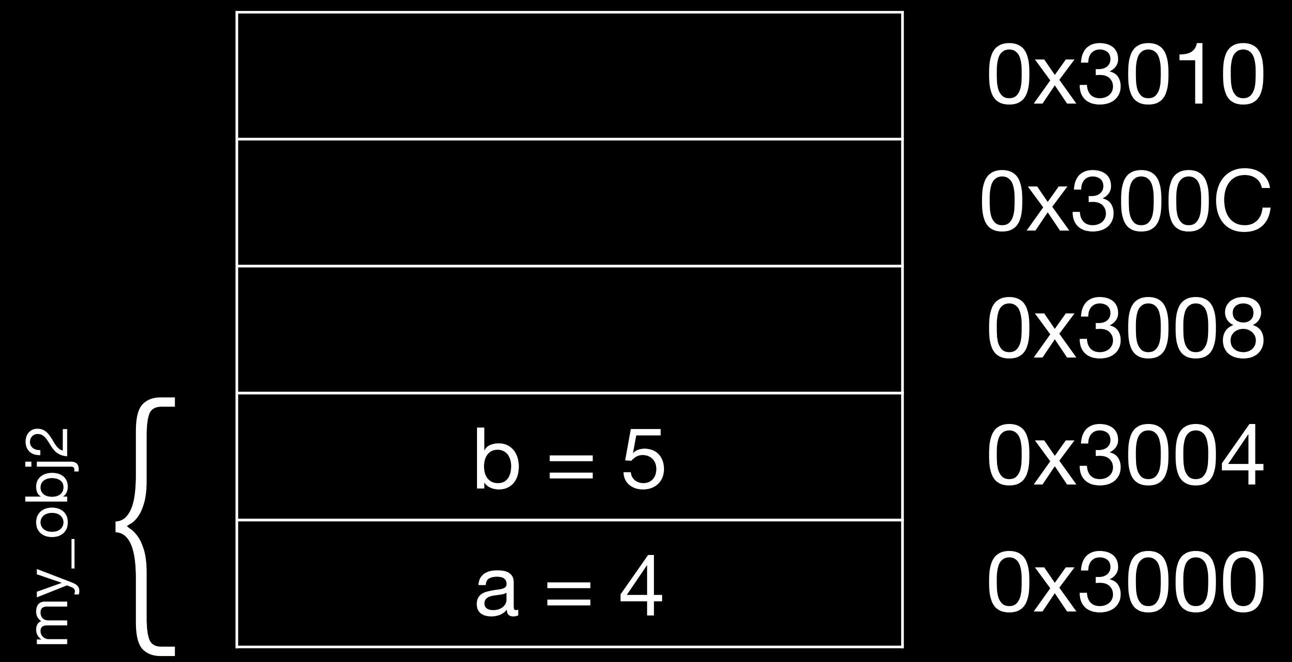


First inheritance

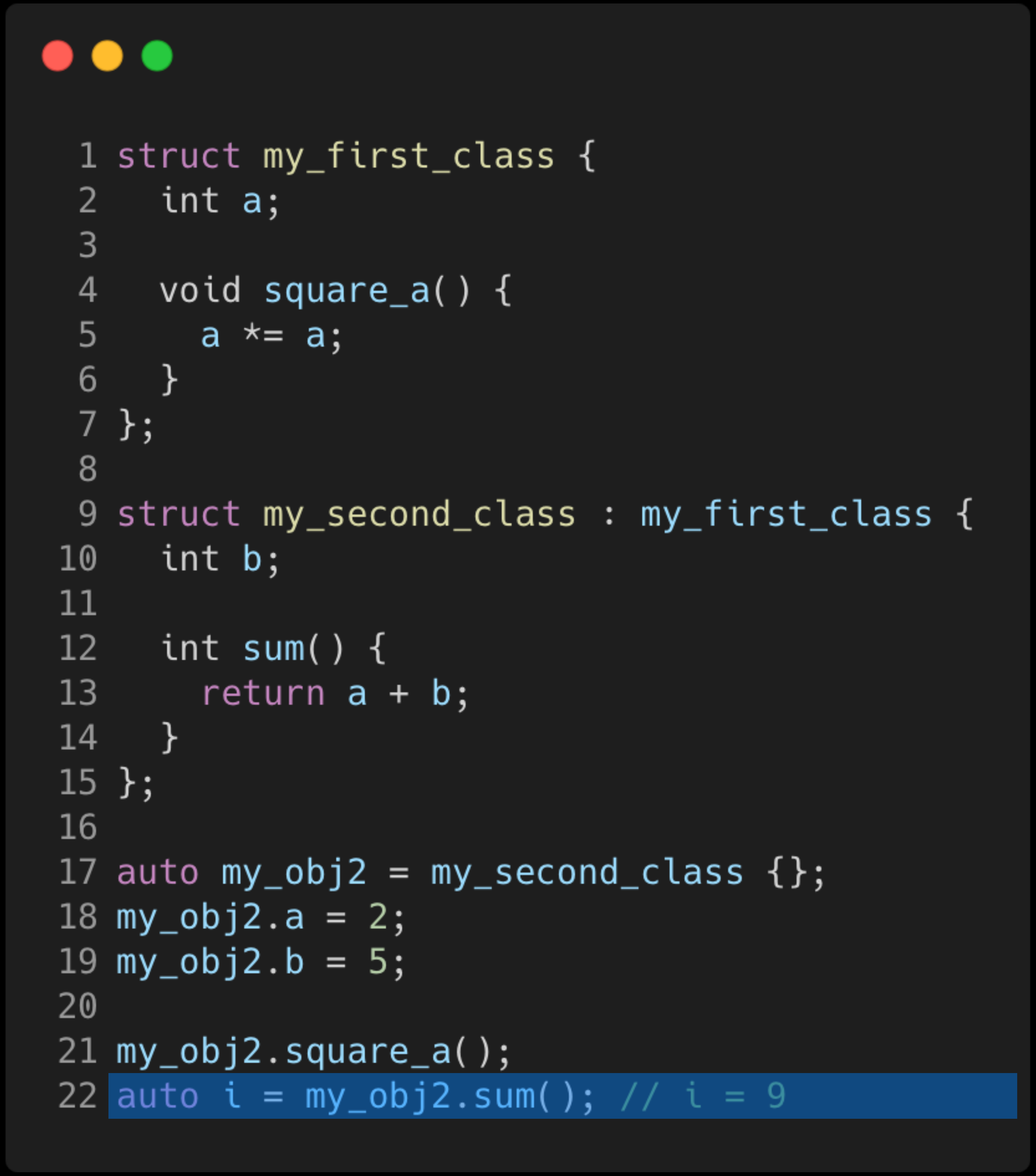


```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

Memory layout

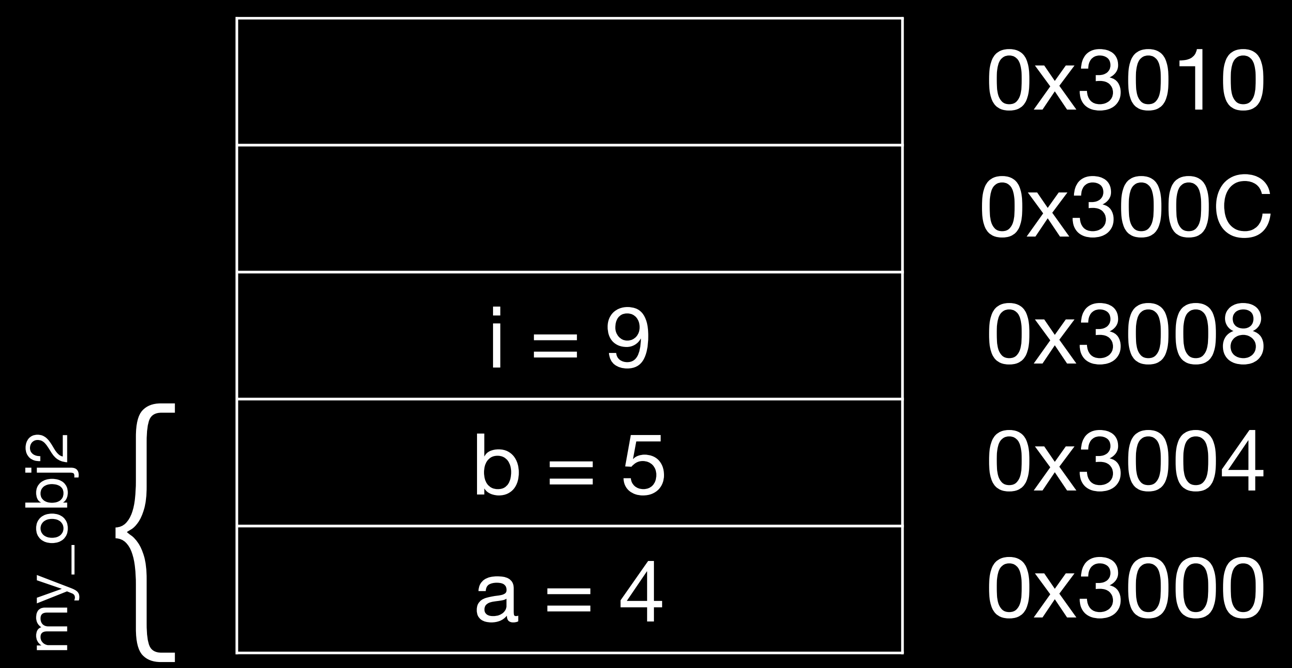


First inheritance



```cpp
struct my_first_class {
  int a;

  void square_a() {
    a *= a;
  }
};

struct my_second_class : my_first_class {
  int b;

  int sum() {
    return a + b;
  }
};

auto my_obj2 = my_second_class {};
my_obj2.a = 2;
my_obj2.b = 5;

my_obj2.square_a();
auto i = my_obj2.sum(); // i = 9
```

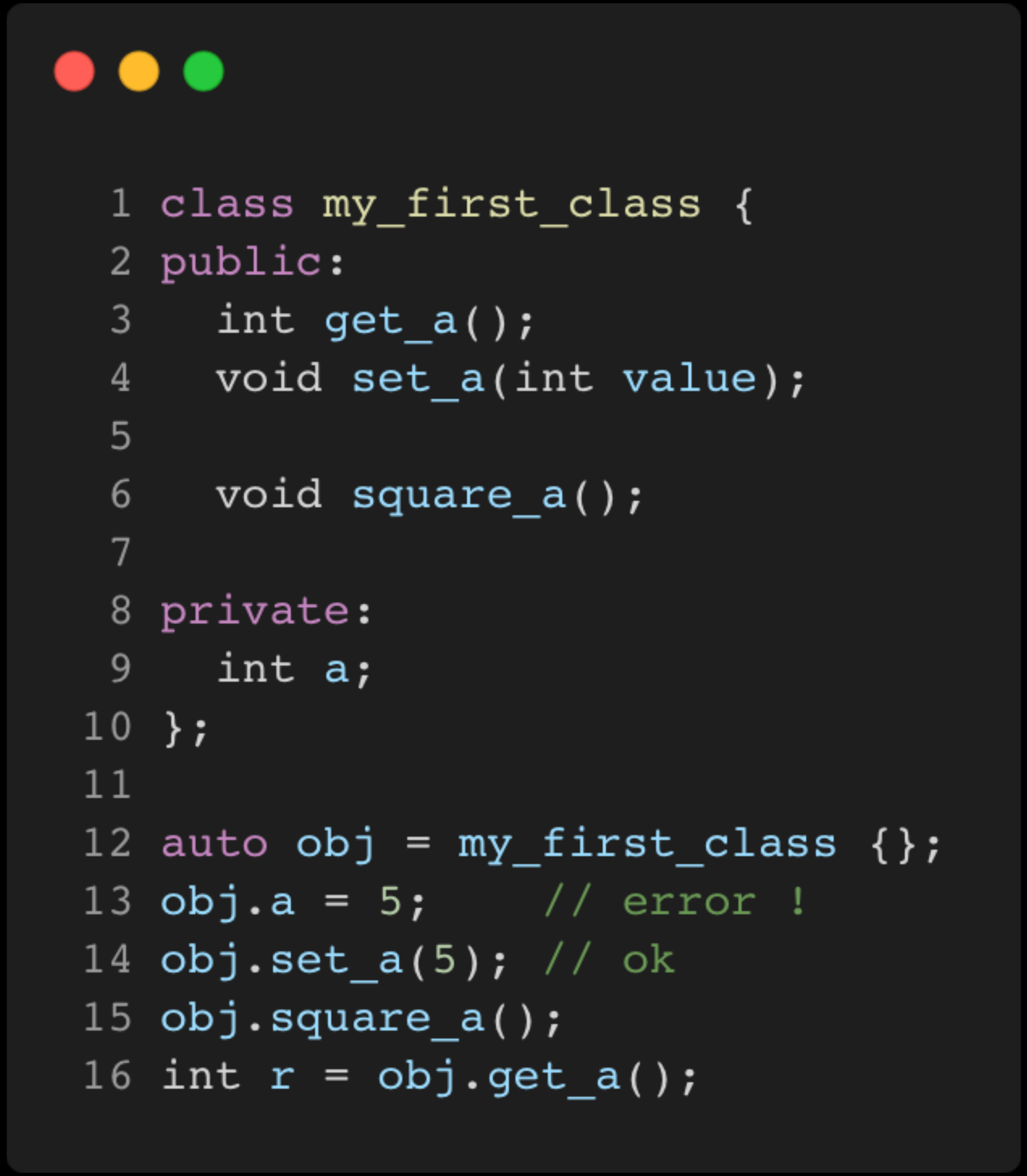
Memory layout



Managing access to class members

public / private keywords

- private allows access only within the class
- public allows access from anywhere
- The default for class is private
- The default for struct is public



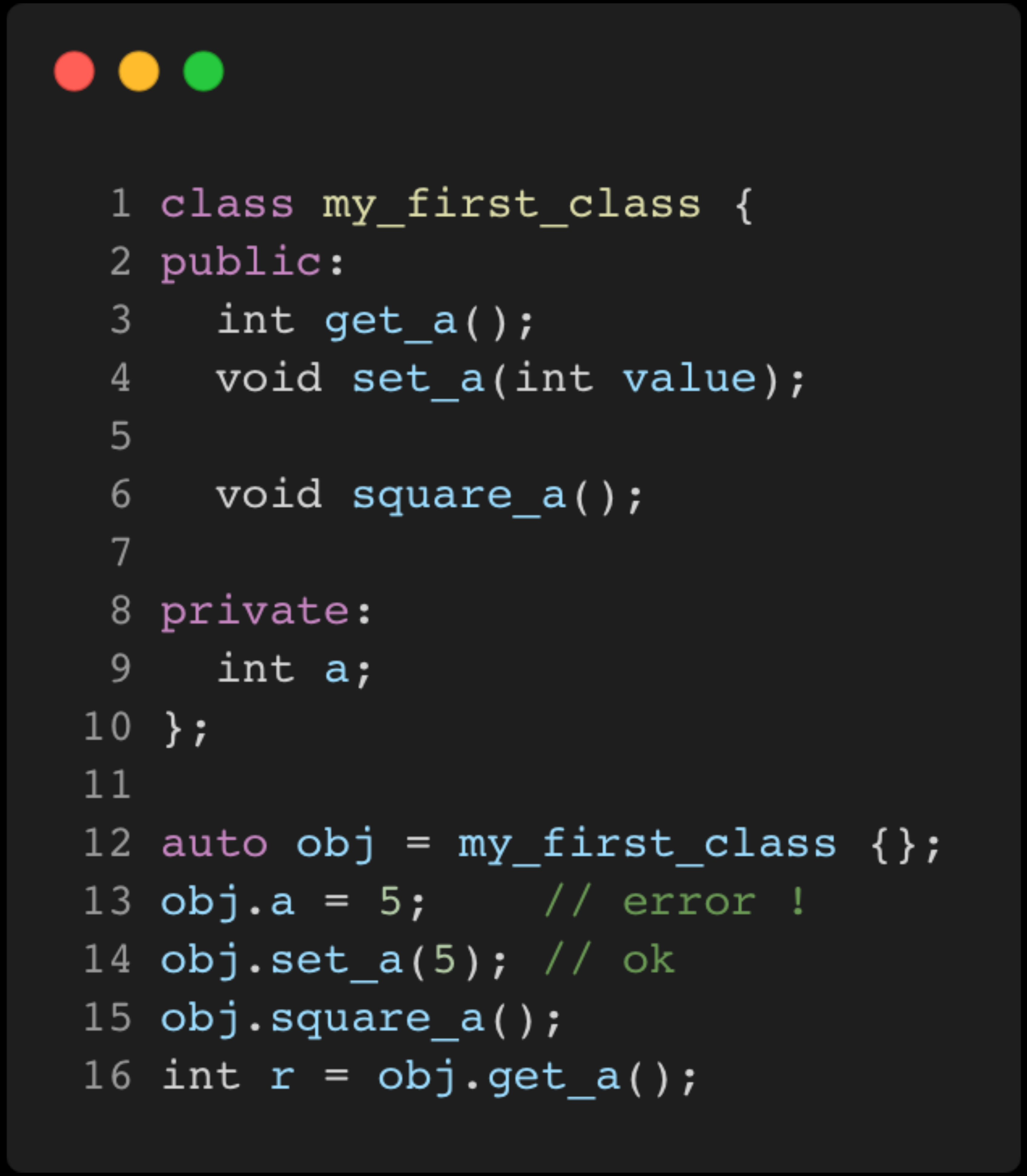
```cpp
class my_first_class {
public:
  int get_a();
  void set_a(int value);

  void square_a();

private:
  int a;
};

auto obj = my_first_class {};
obj.a = 5;     // error !
obj.set_a(5); // ok
obj.square_a();
int r = obj.get_a();
```

Managing access to class members

public / private keywords

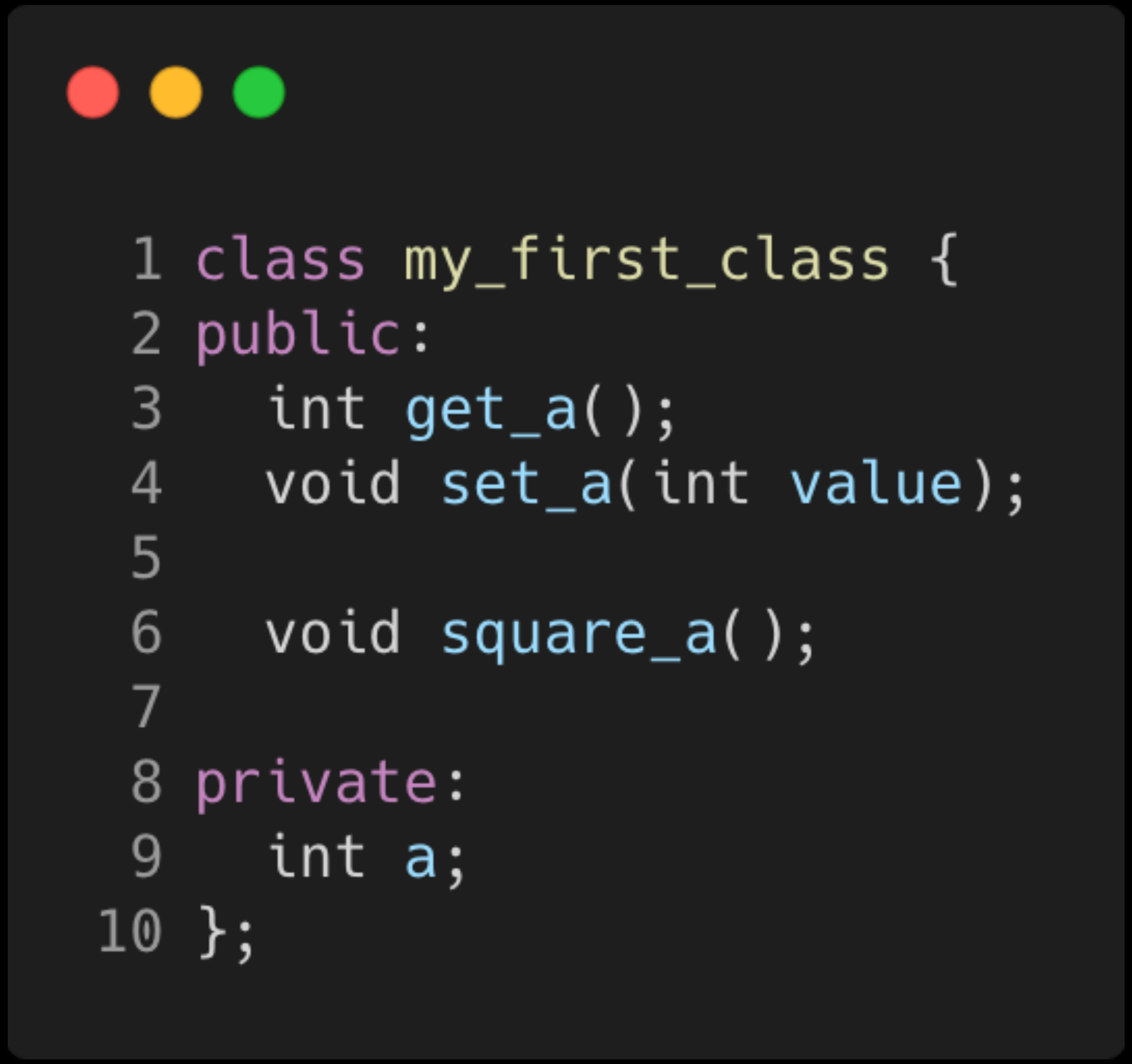
- private allows access only within the class
- public allows access from anywhere
- The default for class is private
- The default for struct is public

This break my_second_class !

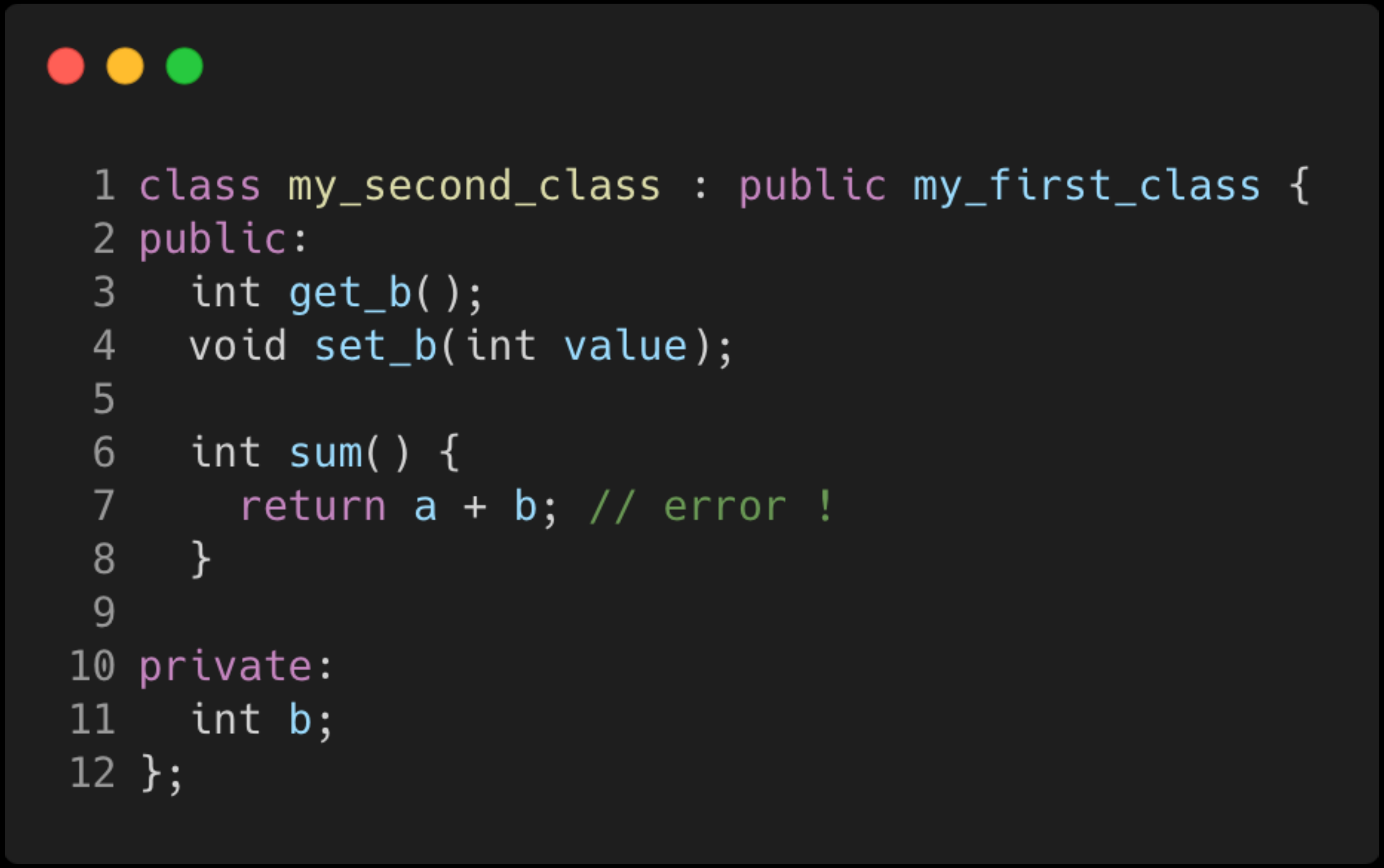


```cpp
class my_first_class {
public:
  int get_a();
  void set_a(int value);

  void square_a();

private:
  int a;
};

auto obj = my_first_class {};
obj.a = 5;     // error !
obj.set_a(5); // ok
obj.square_a();
int r = obj.get_a();
```

a is not accessible in the sum function

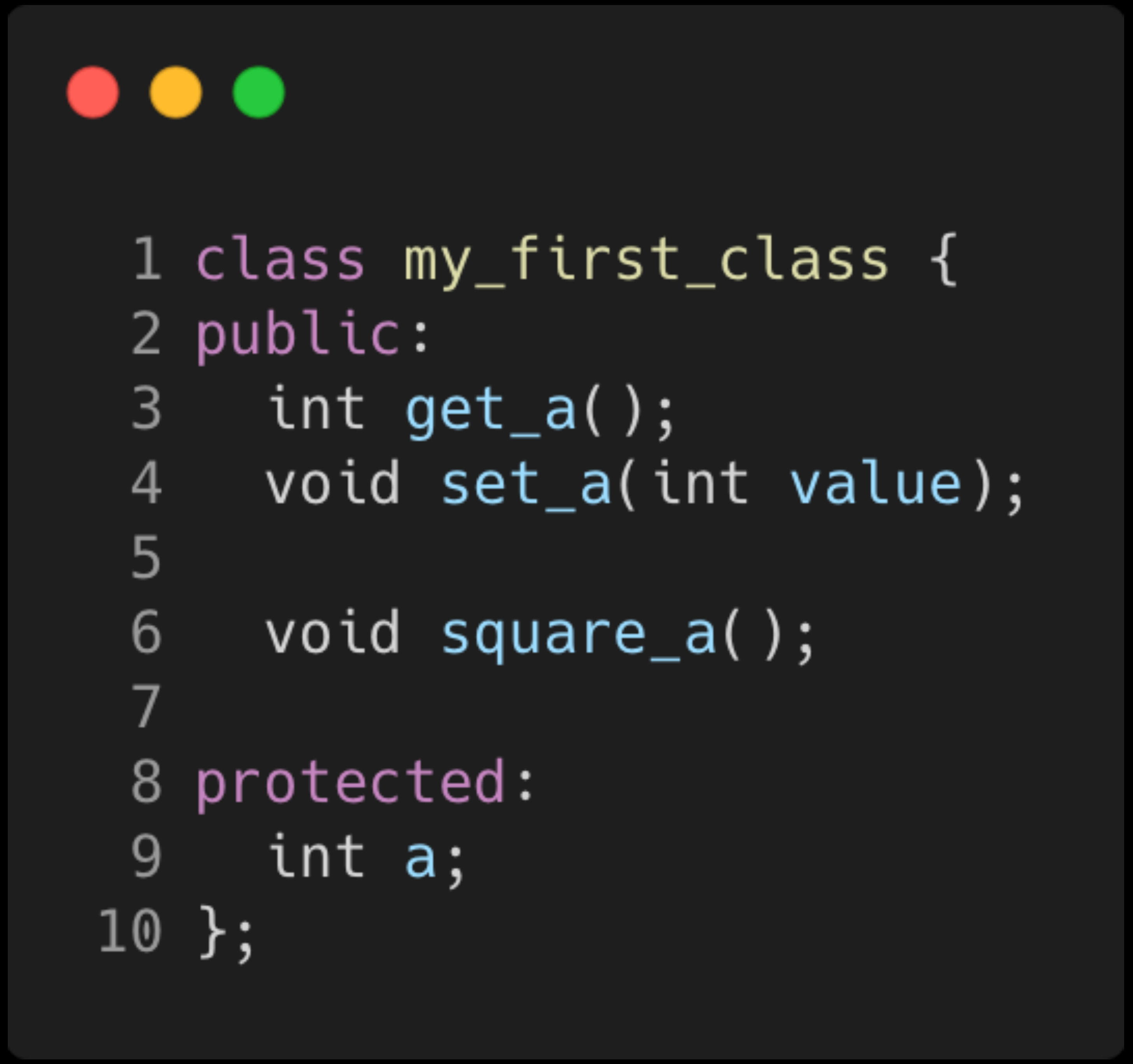


```cpp
class my_first_class {
public:
  int get_a();
  void set_a(int value);

  void square_a();

private:
  int a;
};
```

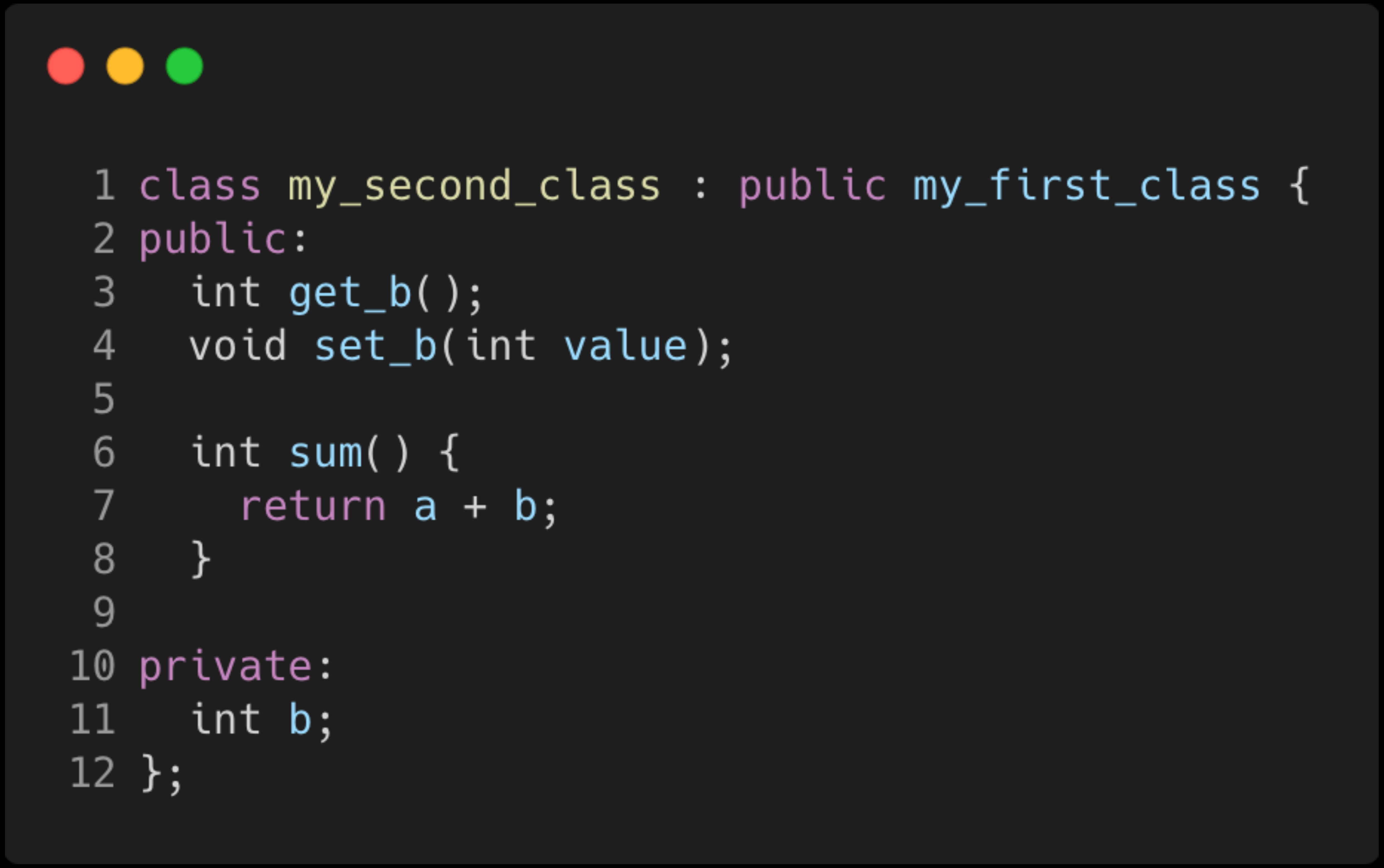


```cpp
class my_second_class : public my_first_class {
public:
  int get_b();
  void set_b(int value);

  int sum() {
    return a + b; // error !
  }

private:
  int b;
};
```

Solution is protected keyword



```cpp
class my_first_class {
public:
  int get_a();
  void set_a(int value);

  void square_a();

protected:
  int a;
};
```



```cpp
class my_second_class : public my_first_class {
public:
  int get_b();
  void set_b(int value);

  int sum() {
    return a + b;
  }

private:
  int b;
};
```

Inheritance can be public, protected or private

It influences the privacy of inherited members for external code.

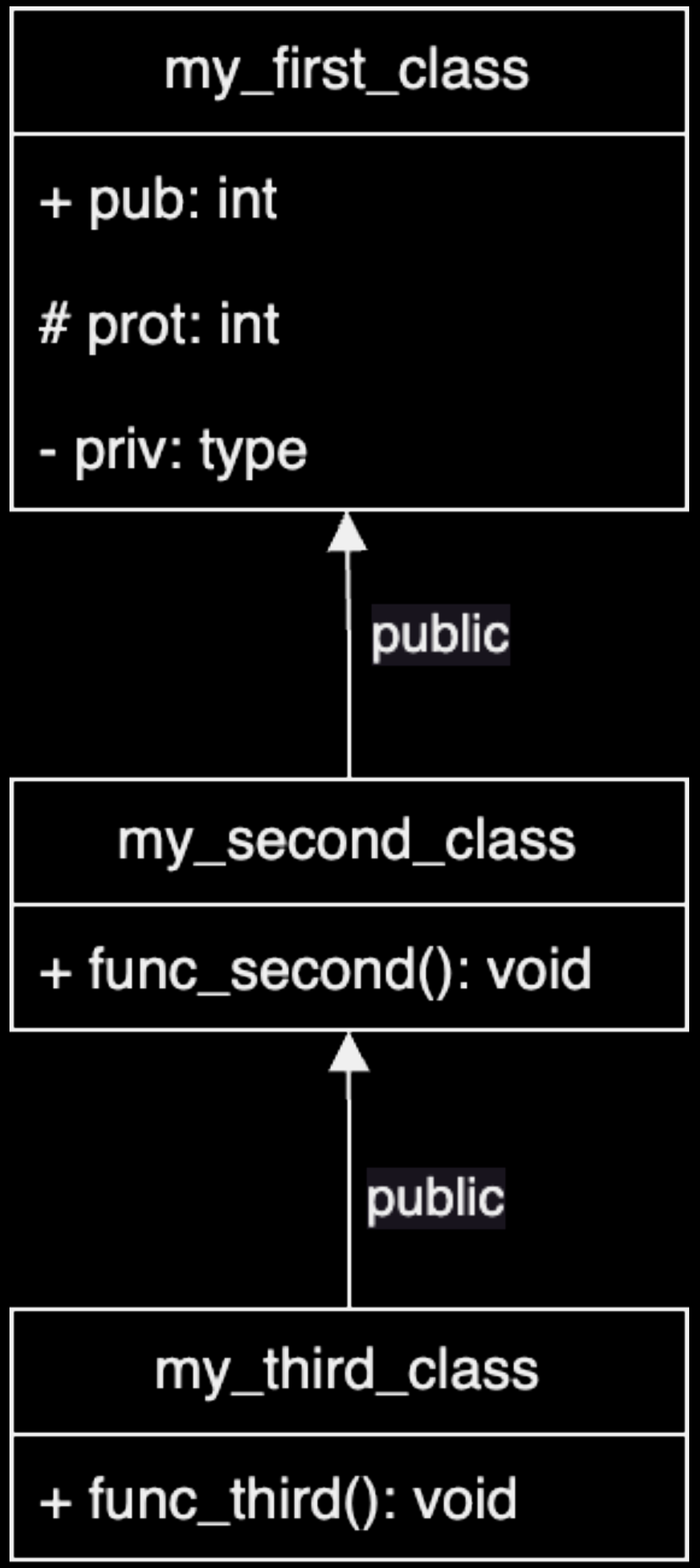
The code of the class itself is not affected

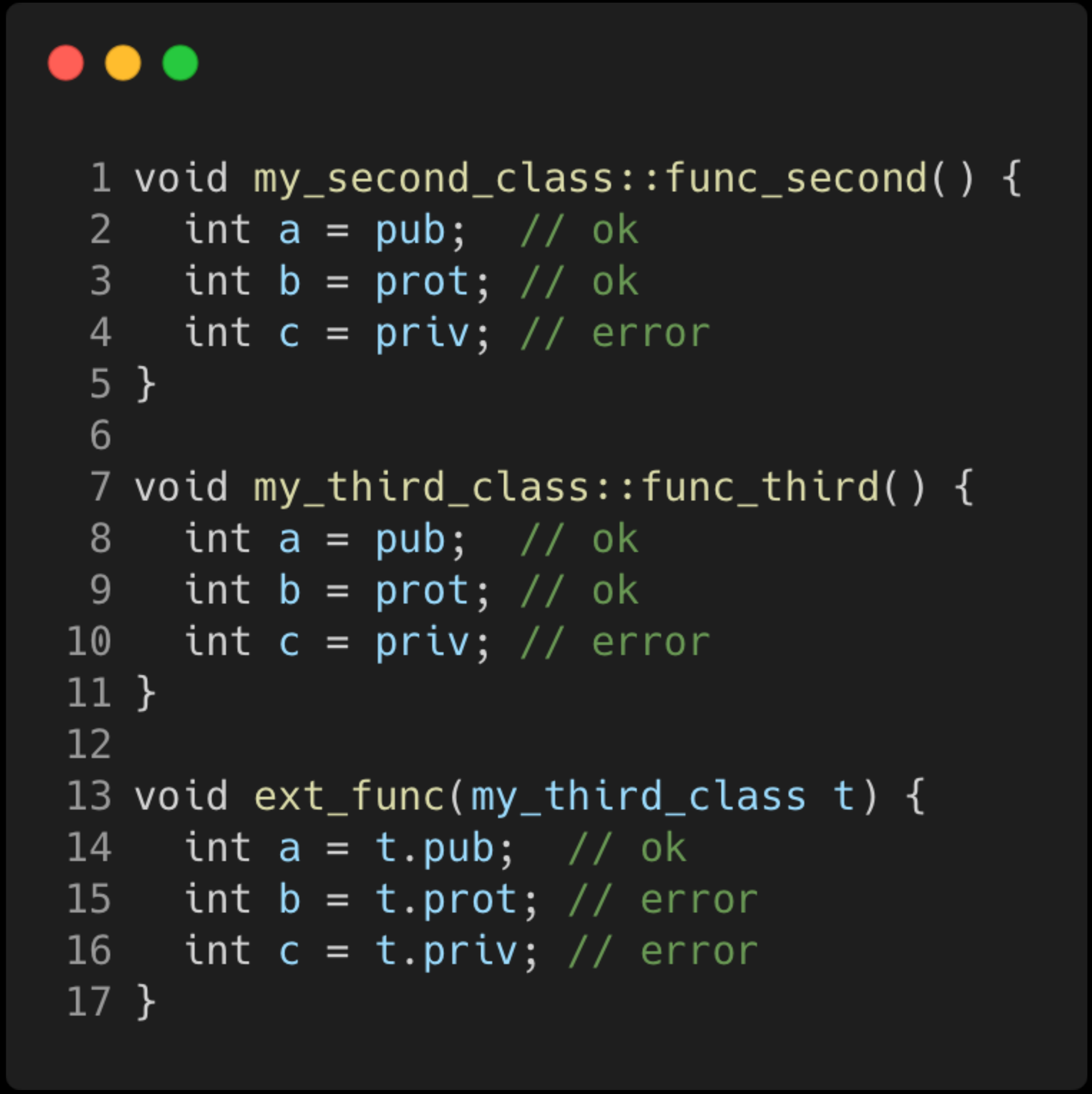
- public privacy of inherited members remains unchanged
- protected inherited public members are seen as protected
- private all inherited members are seen as private. This is the default for classes if nothing is specified

Inheritance can be public, protected or private

- Net result for external code
    - only public members of public inheritance are accessible
- Net result for code in derived classes
    - only public and protected members of public and protected
    - parents are accessible

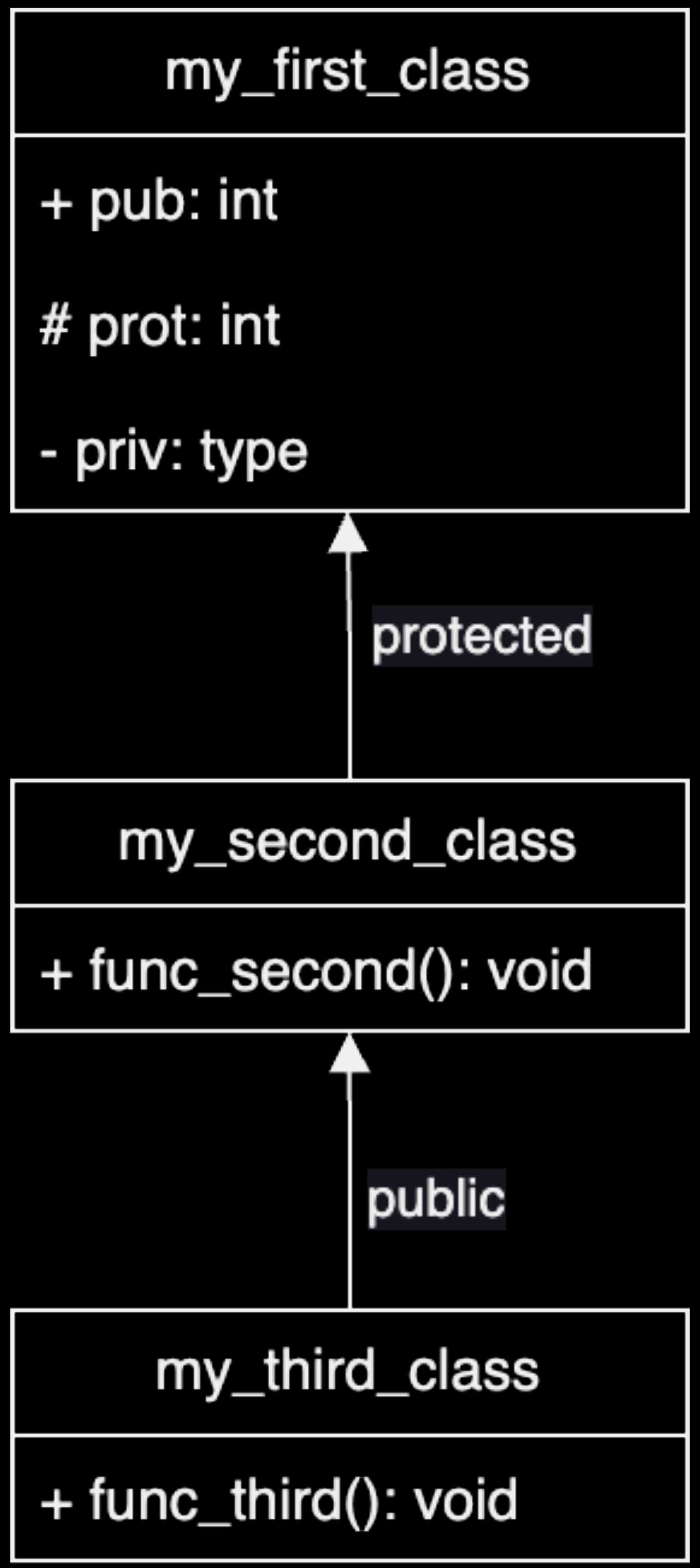
Managing inheritance privacy - public

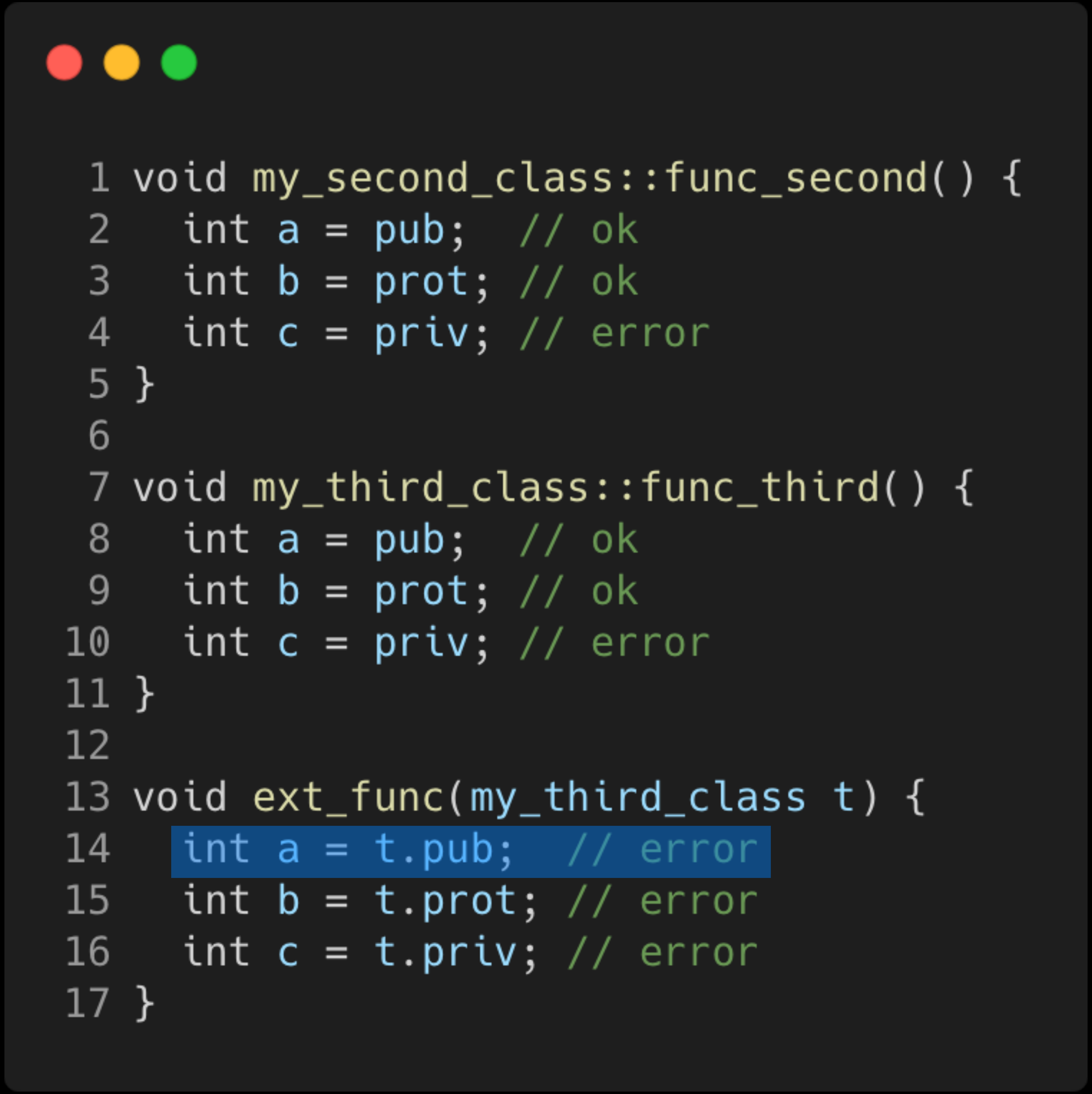




```
void my_second_class::func_second() {
  int a = pub;  // ok
  int b = prot; // ok
  int c = priv; // error
}

void my_third_class::func_third() {
  int a = pub;  // ok
  int b = prot; // ok
  int c = priv; // error
}

void ext_func(my_third_class t) {
  int a = t.pub;  // ok
  int b = t.prot; // error
  int c = t.priv; // error
}
```

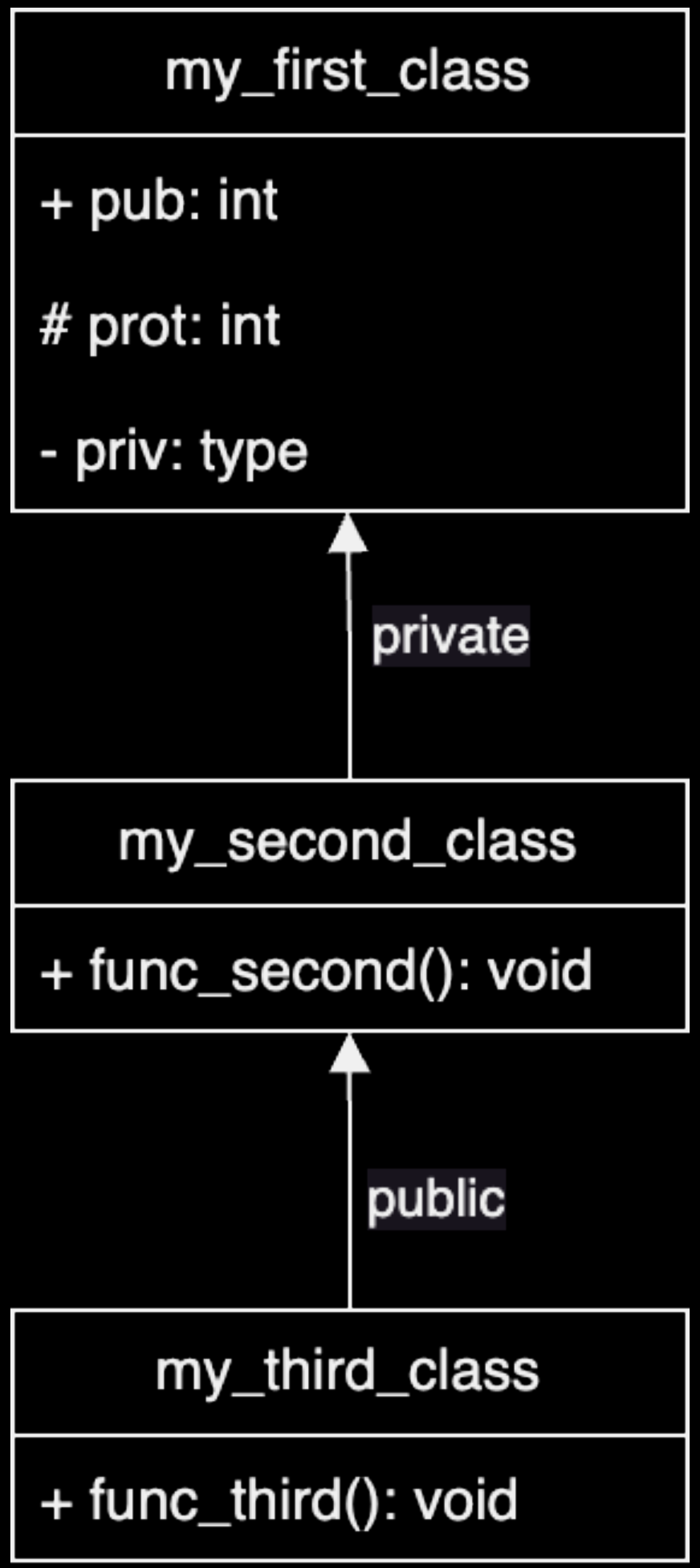
Managing inheritance privacy - protected

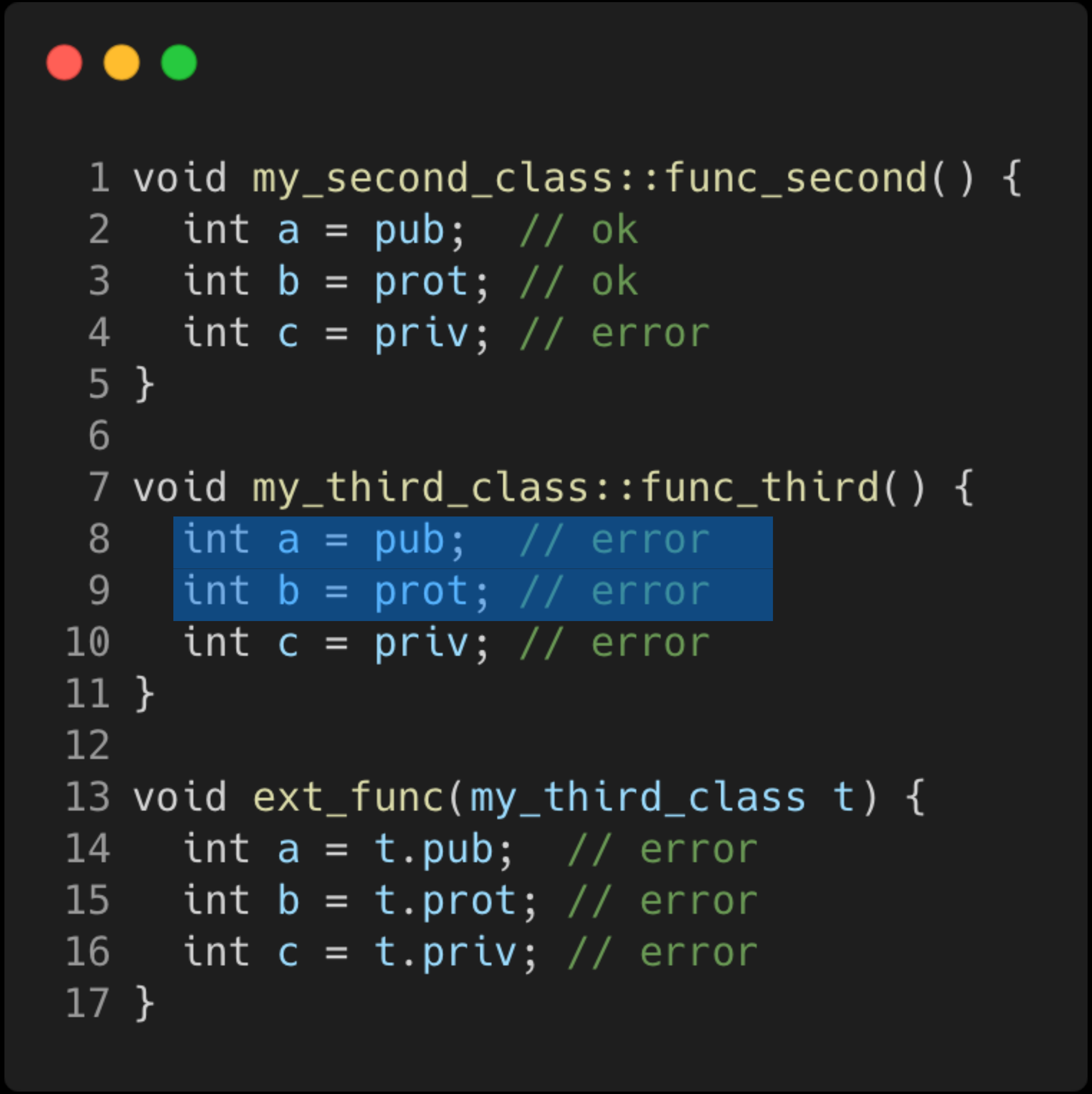




```
void my_second_class::func_second() {
  int a = pub;  // ok
  int b = prot; // ok
  int c = priv; // error
}

void my_third_class::func_third() {
  int a = pub;  // ok
  int b = prot; // ok
  int c = priv; // error
}

void ext_func(my_third_class t) {
  int a = t.pub;  // error
  int b = t.prot; // error
  int c = t.priv; // error
}
```

Managing inheritance privacy - private



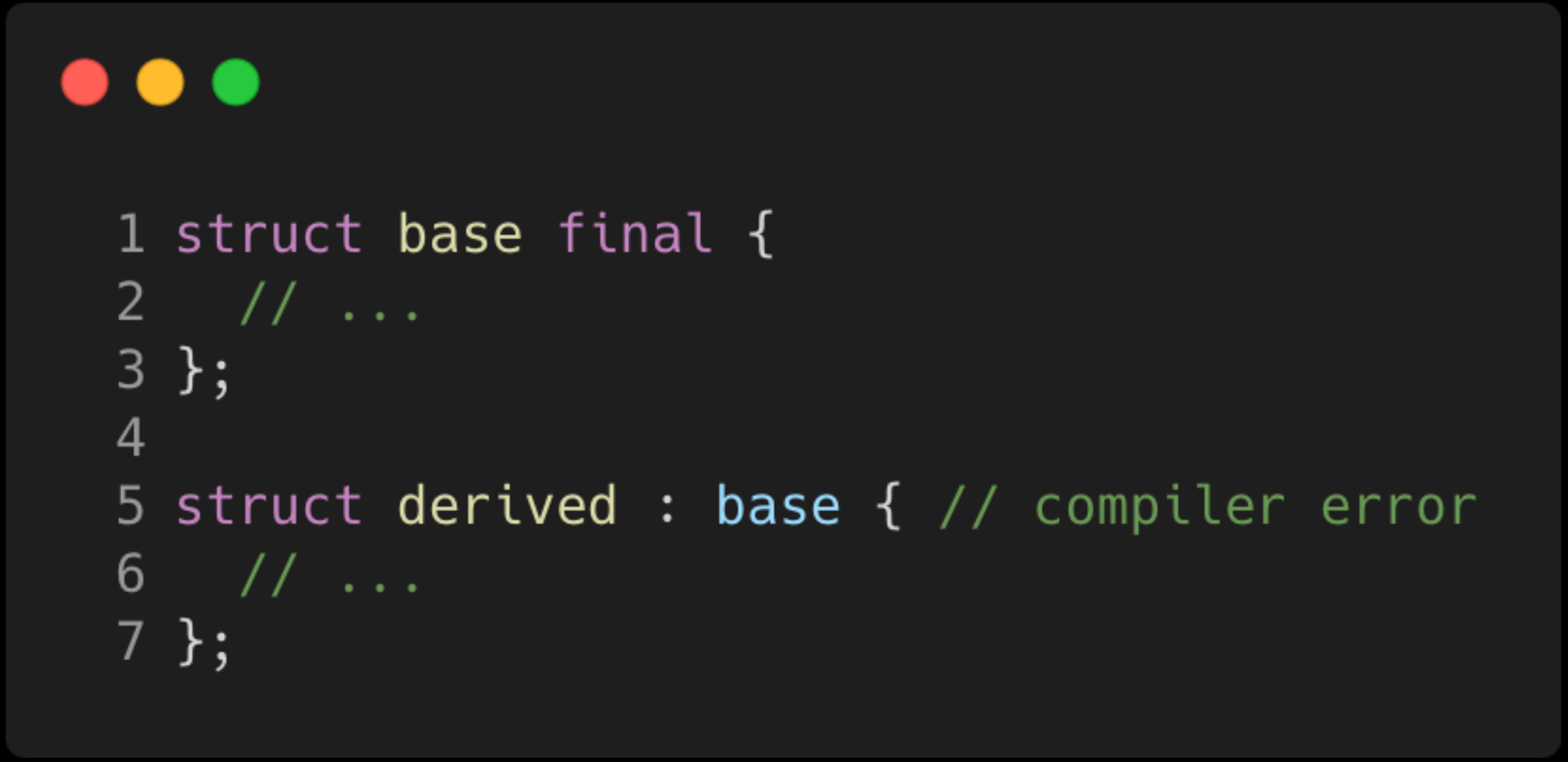


```cpp
void my_second_class::func_second() {
  int a = pub;  // ok
  int b = prot; // ok
  int c = priv; // error
}

void my_third_class::func_third() {
  int a = pub;  // error
  int b = prot; // error
  int c = priv; // error
}

void ext_func(my_third_class t) {
  int a = t.pub;  // error
  int b = t.prot; // error
  int c = t.priv; // error
}
```

Final class

Idea

- make sure you cannot inherit from a given class
- by declaring it final



```cpp
struct base final {
  // ...
};

struct derived : base { // compiler error
  // ...
};
```

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

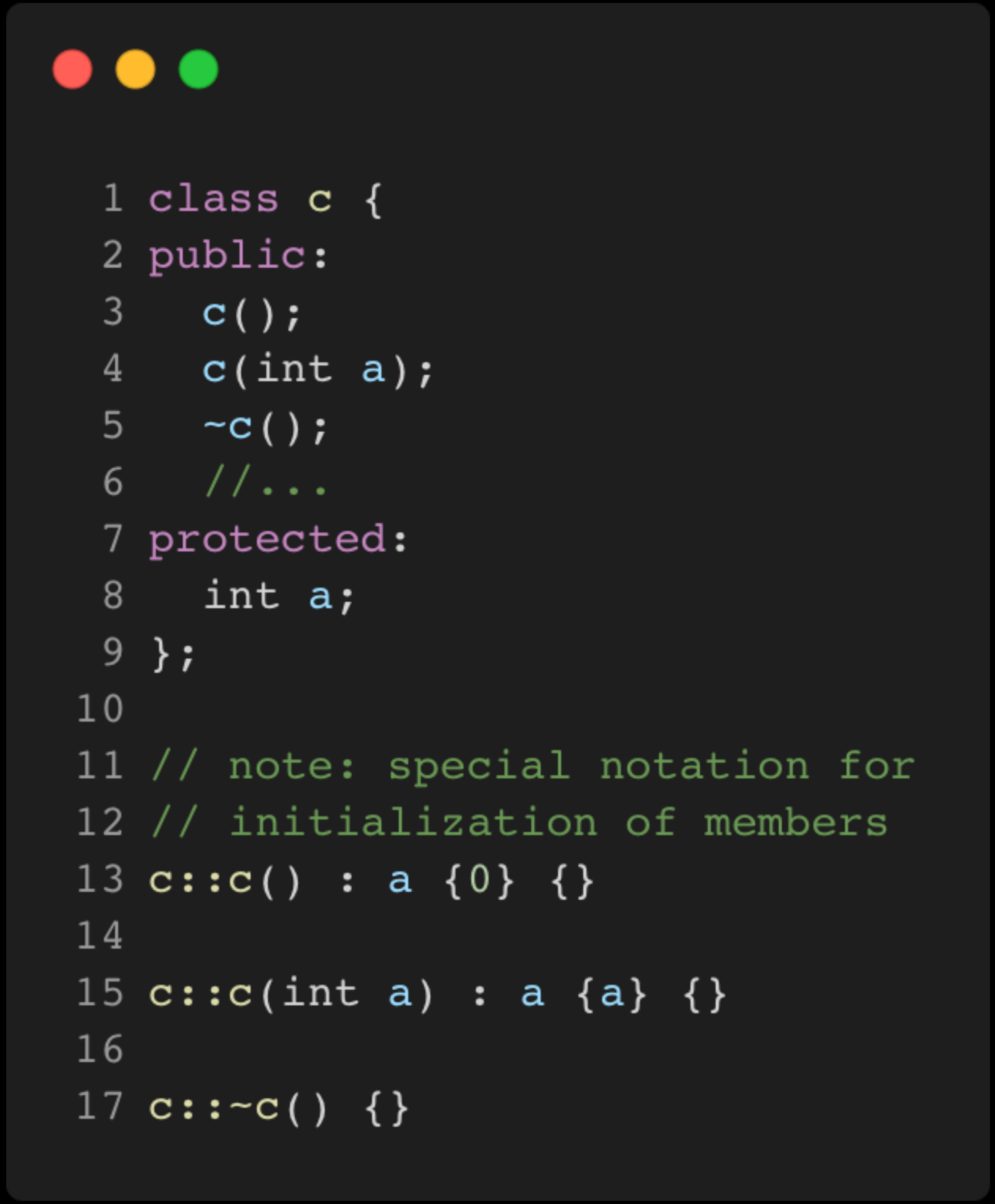
Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Class constructors and destructors

Concept

- special functions called when building/destroying an object
- a class can have several constructors, but only one destructor
- the constructors have the same name as the class
- same for the destructor with a leading ~



```cpp
class c {
public:
  c();
  c(int a);
  ~c();
  //...
protected:
  int a;
};

// note: special notation for
// initialization of members
c::c() : a {0} {}

c::c(int a) : a {a} {}

c::~c() {}
```

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Objects Oriented Programming (OOP)

- Objects and classes
- Inheritance
- Constructors / Destructors
- Static members
- Allocating objects
- Advanced Object Oriented
- Type casing
- Operator overloading
- Function objects
- Name Lookups

Outline

End