THE PROGRAMMING CLUB PRESENTS

PC1020

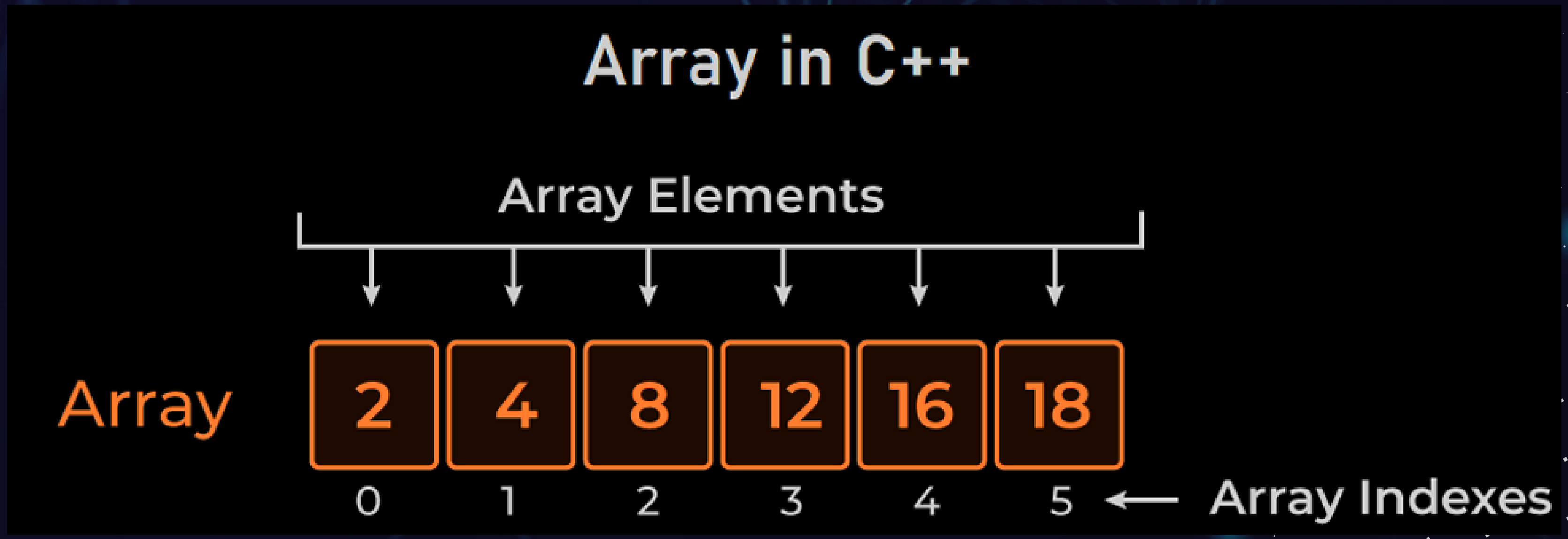
INTRO TO PROGRAMMING

“IMAGINE YOUR PROF IS KEEPING TRACK OF THE MARKS OF 100 STUDENTS IN AN APP. DO YOU THINK THE APP CREATES 100 SEPARATE VARIABLES FOR EACH EXAM, OR SOMETHING BETTER?”

ARRAYS

(FIXED SIZE CONTAINERS)

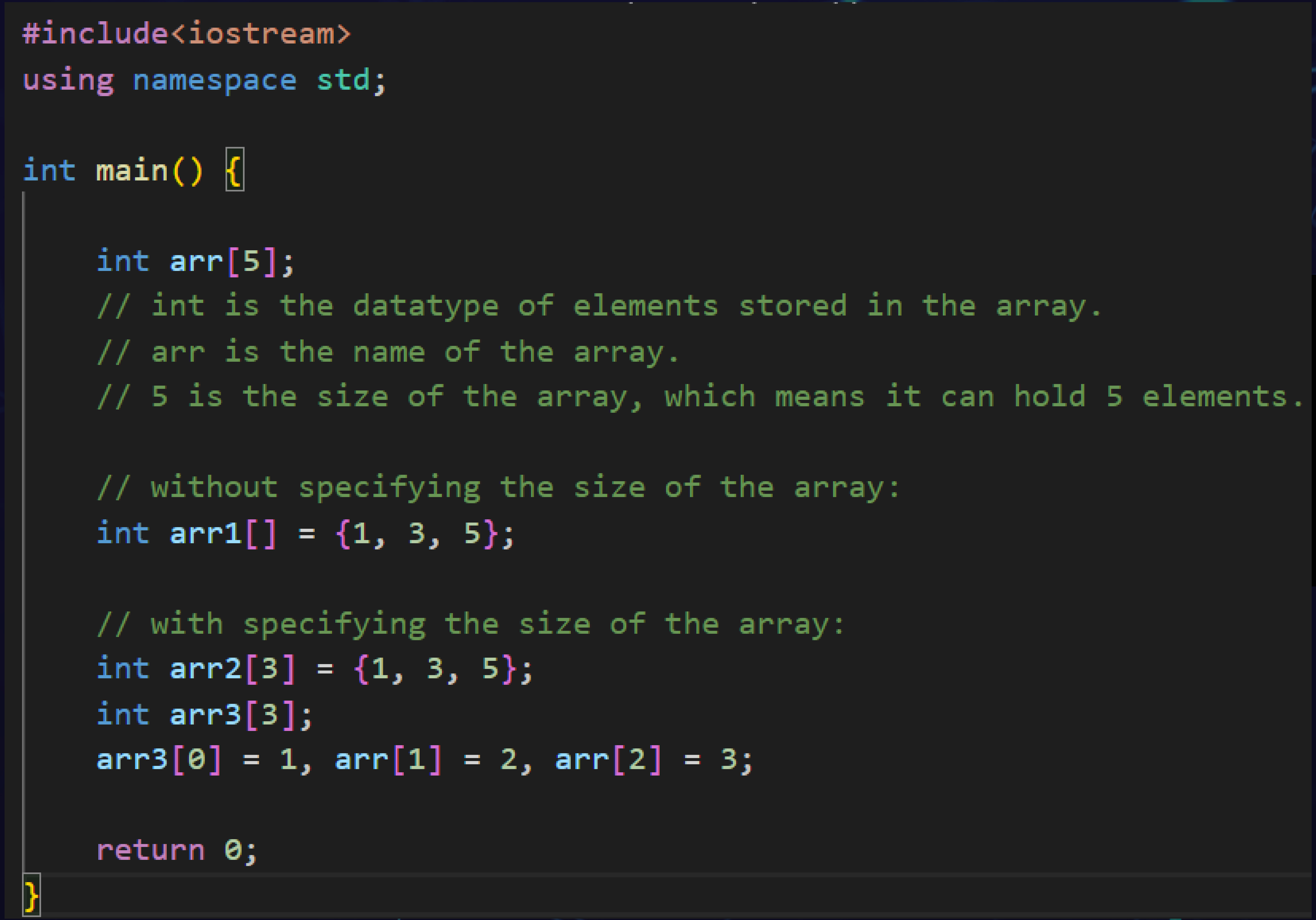
- INDEXING OF AN ARRAY STARTS FROM 0.
- ONCE AN ARRAY IS DECLARED ITS SIZE REMAINS CONSTANT THROUGHOUT THE PROGRAM.



SO HOW DO WE DECLARE AN ARRAY AND HOW DO WE USE IT !!?

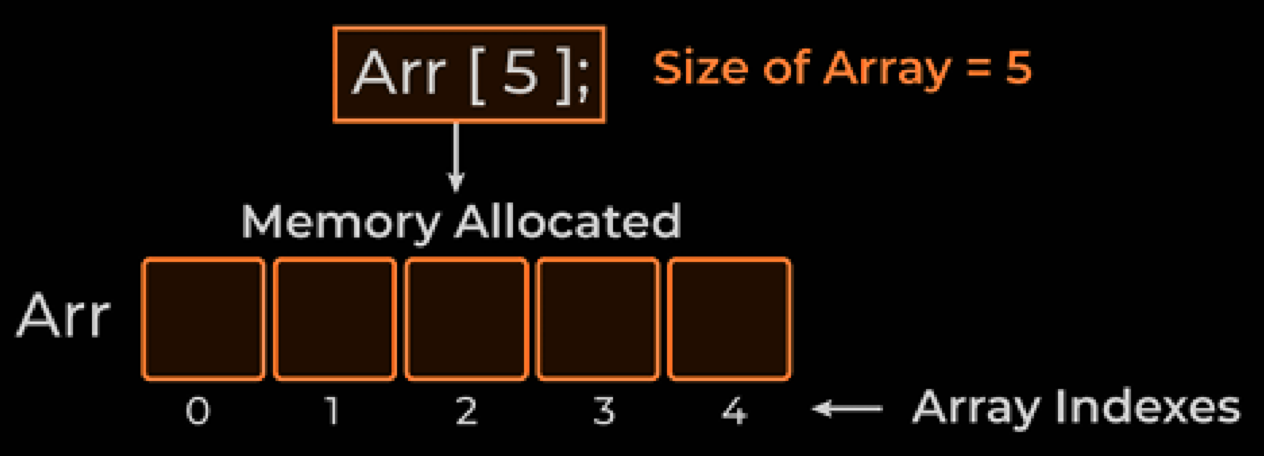
ARRAYS

(FIXED SIZE CONTAINERS)



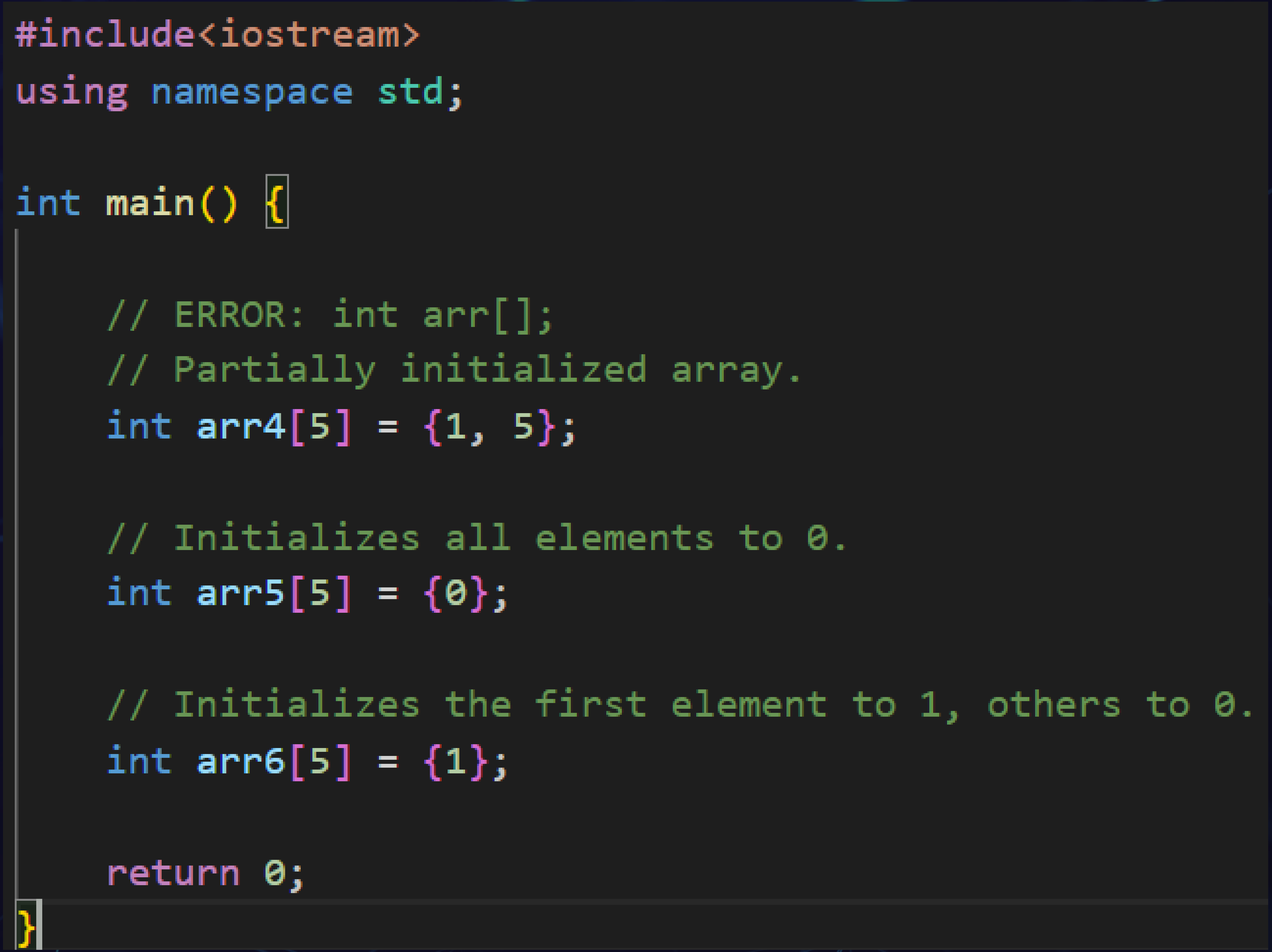
```cpp
#include<iostream>
using namespace std;

int main() {

    int arr[5];
    // int is the datatype of elements stored in the array.
    // arr is the name of the array.
    // 5 is the size of the array, which means it can hold 5 elements.

    // without specifying the size of the array:
    int arr1[] = {1, 3, 5};

    // with specifying the size of the array:
    int arr2[3] = {1, 3, 5};
    int arr3[3];
    arr3[0] = 1, arr[1] = 2, arr[2] = 3;

    return 0;
}
```

Array Declaration



ARRAYS

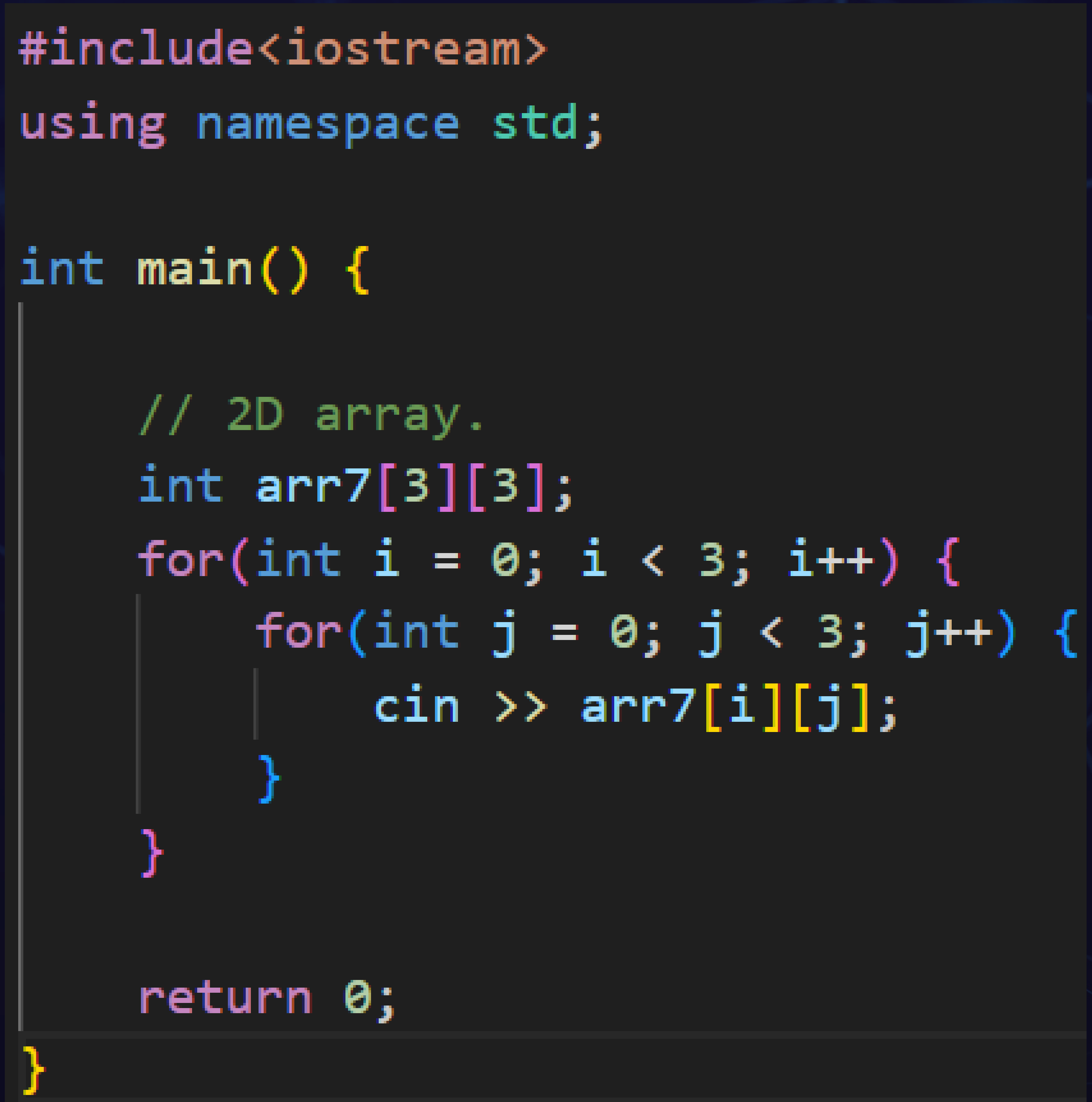
(FIXED SIZE CONTAINERS)



```cpp
#include<iostream>
using namespace std;

int main() {

    // ERROR: int arr[];
    // Partially initialized array.
    int arr4[5] = {1, 5};

    // Initializes all elements to 0.
    int arr5[5] = {0};

    // Initializes the first element to 1, others to 0.
    int arr6[5] = {1};

    return 0;
}
```

ARRAYS

(FIXED SIZE CONTAINERS)



```cpp
#include<iostream>
using namespace std;

int main() {

    // 2D array.
    int arr7[3][3];
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            cin >> arr7[i][j];
        }
    }

    return 0;
}
```

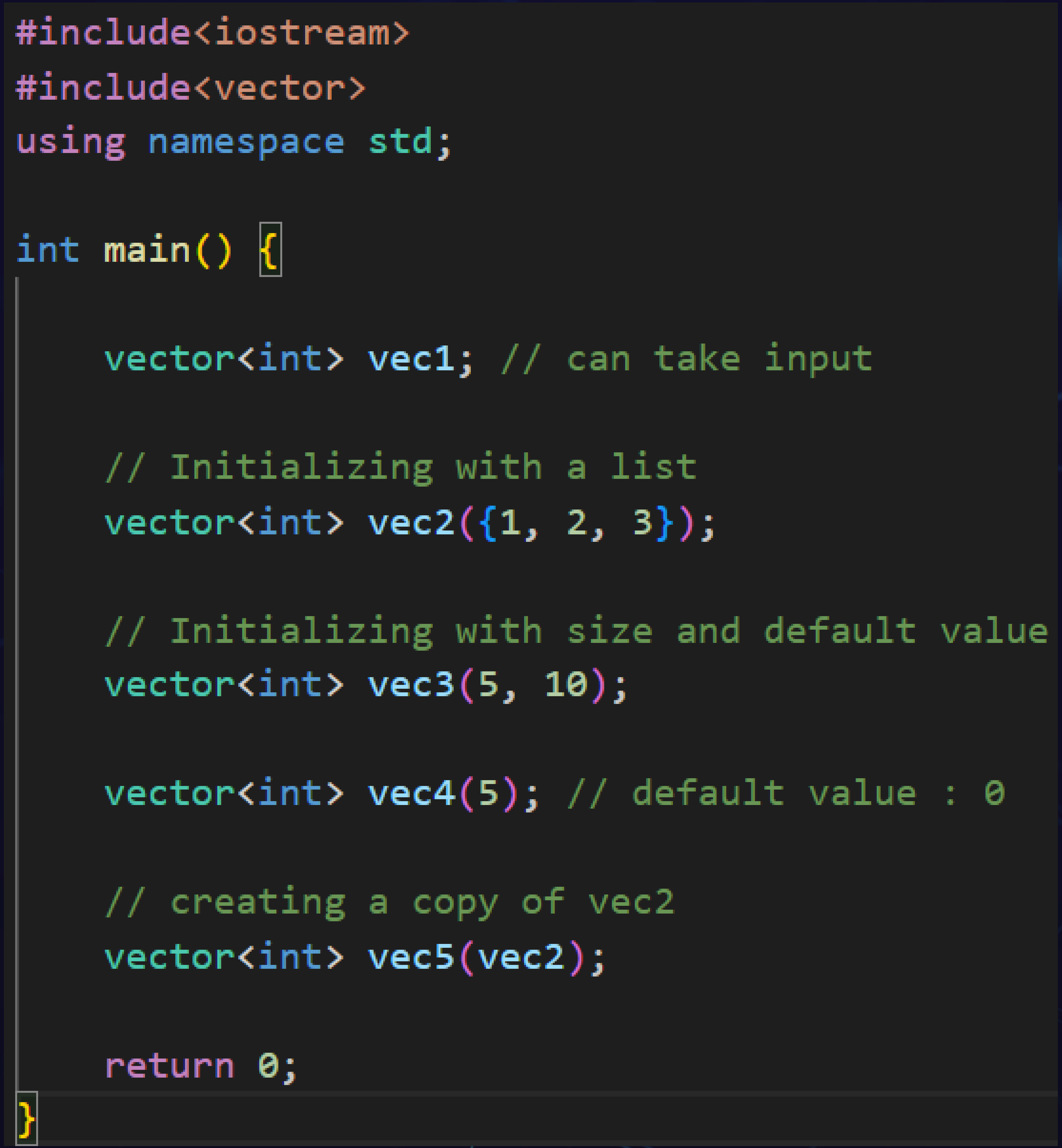
	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

“WHAT IF WE DON’T KNOW THE NUMBER OF STUDENTS BEFOREHAND?”

VECTORS

(VARIABLE SIZE CONTAINERS)

- Dynamic arrays with the ability to resize themselves, their storage being handled automatically by the container.
- std::vector in C++ is the class template that contains the vector container and its member functions. It is defined inside the <vector> header file.
- Syntax ==> std::vector<datatype> vectorname;

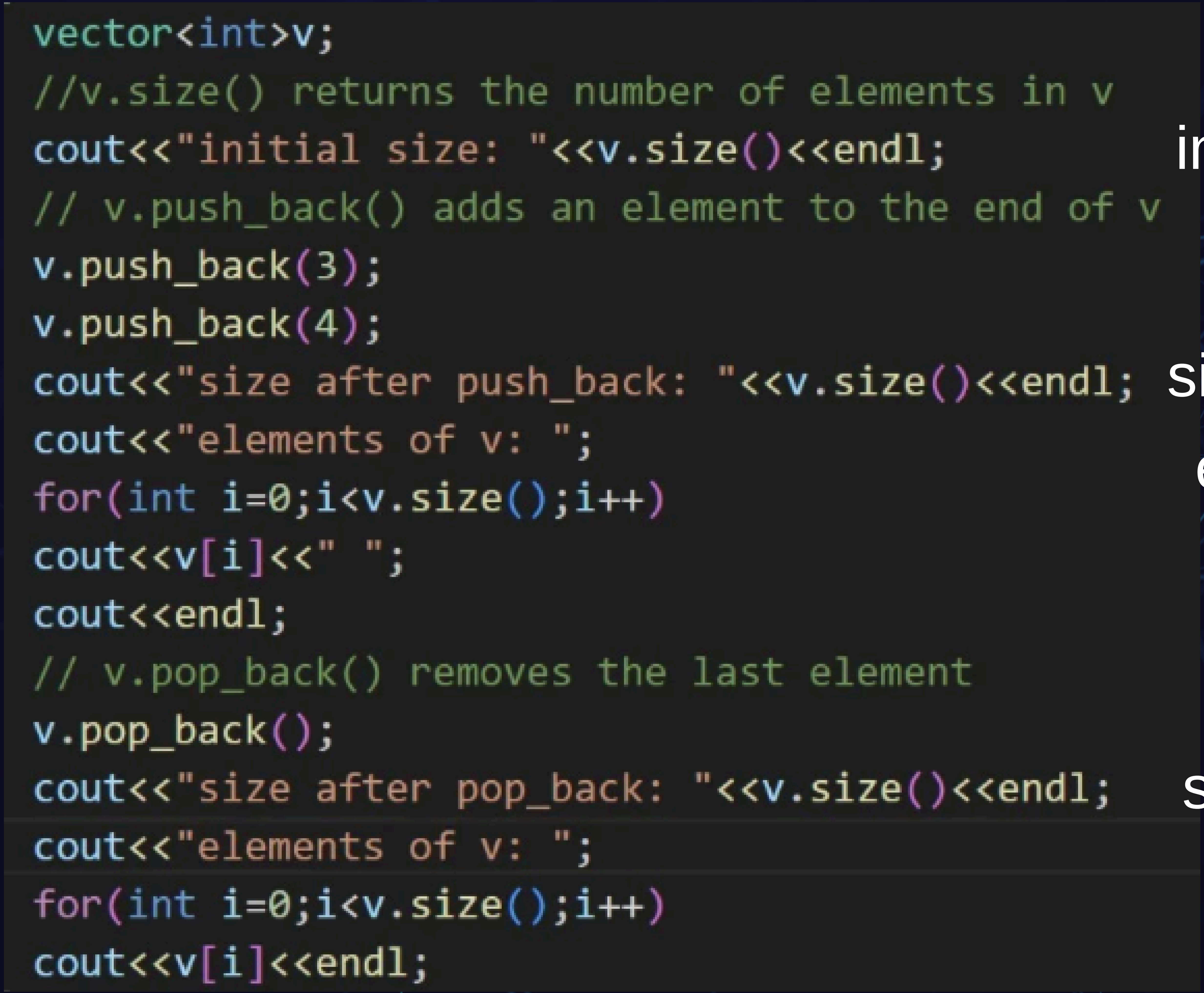


```cpp
#include<iostream>
#include<vector>
using namespace std;

int main() {

    vector<int> vec1; // can take input

    // Initializing with a list
    vector<int> vec2({1, 2, 3});

    // Initializing with size and default value
    vector<int> vec3(5, 10);

    vector<int> vec4(5); // default value : 0

    // creating a copy of vec2
    vector<int> vec5(vec2);

    return 0;
}
```

```
size: 0, capacity: 0

[1, 2, 3]

[10, 10, 10, 10, 10]
[0, 0, 0, 0, 0]

[1, 2, 3]
```

COMMON MEMBER FUNCTIONS

- **vec.push_back(3)** : pushes 3 to the end of vector.
- **vec.pop_back()** : pops the element at the end.
- **vec.size()** : returns size of vector.
- **vec.empty()** : bool value which is True(1) if vector is empty.



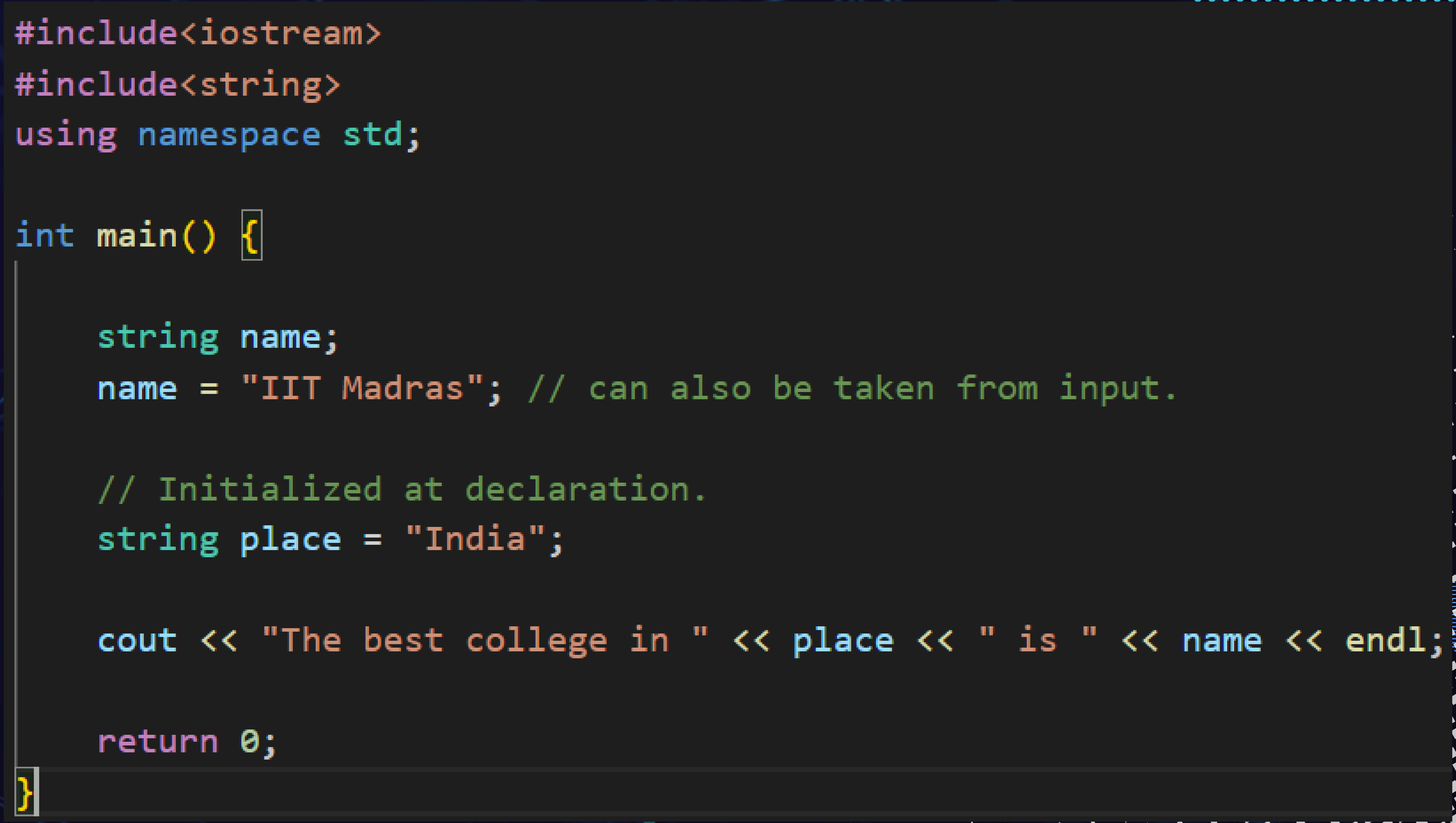
```
vector<int>v;
//v.size() returns the number of elements in v
cout<<"initial size: "<<v.size()<<endl;
// v.push_back() adds an element to the end of v
v.push_back(3);
v.push_back(4);
cout<<"size after push_back: "<<v.size()<<endl;
cout<<"elements of v: ";
for(int i=0;i<v.size();i++)
cout<<v[i]<<" ";
cout<<endl;
// v.pop_back() removes the last element
v.pop_back();
cout<<"size after pop_back: "<<v.size()<<endl;
cout<<"elements of v: ";
for(int i=0;i<v.size();i++)
cout<<v[i]<<endl;
```

```
initial size: 0

size after push_back: 2
elements of v: 3 4

size after pop_back: 1
elements of v: 3
```

“NOW, HOW ABOUT STORING YOUR NAME ALONG WITH YOUR MARKS?”

STRINGS

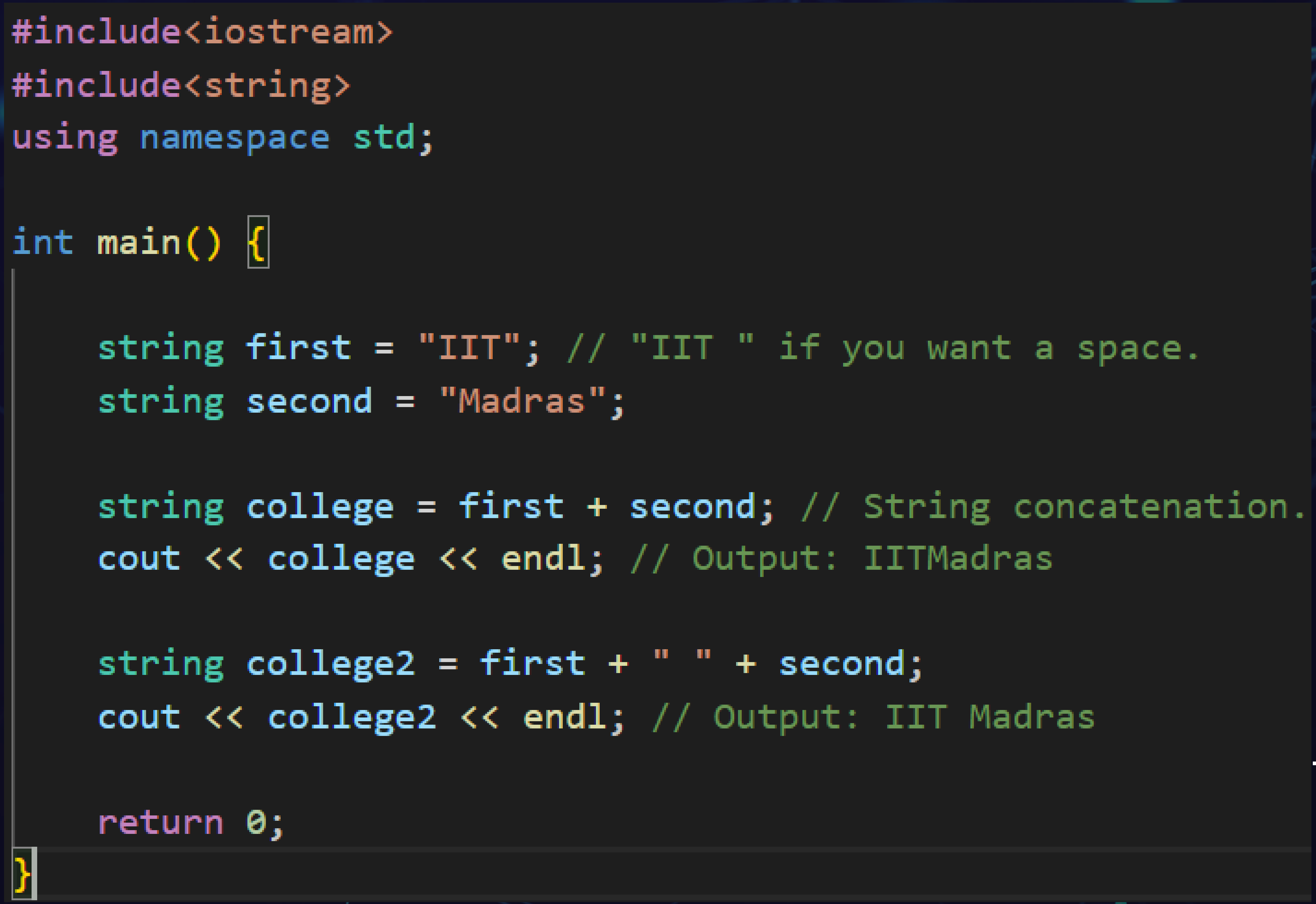
(1D ARRAY OF CHARACTERS)



```cpp
#include<iostream>
#include<string>
using namespace std;

int main() {

    string name;
    name = "IIT Madras"; // can also be taken from input.

    // Initialized at declaration.
    string place = "India";

    cout << "The best college in " << place << " is " << name << endl;

    return 0;
}
```

STRINGS

(1D ARRAY OF CHARACTERS)



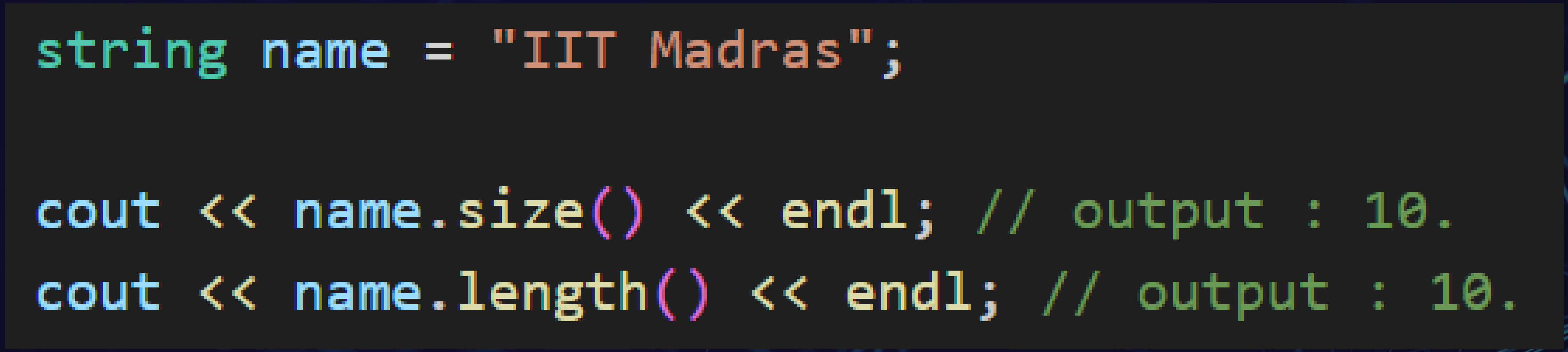
```cpp
#include<iostream>
#include<string>
using namespace std;

int main() {

    string first = "IIT"; // "IIT " if you want a space.
    string second = "Madras";

    string college = first + second; // String concatenation.
    cout << college << endl; // Output: IITMadras

    string college2 = first + " " + second;
    cout << college2 << endl; // Output: IIT Madras

    return 0;
}
```

TIY

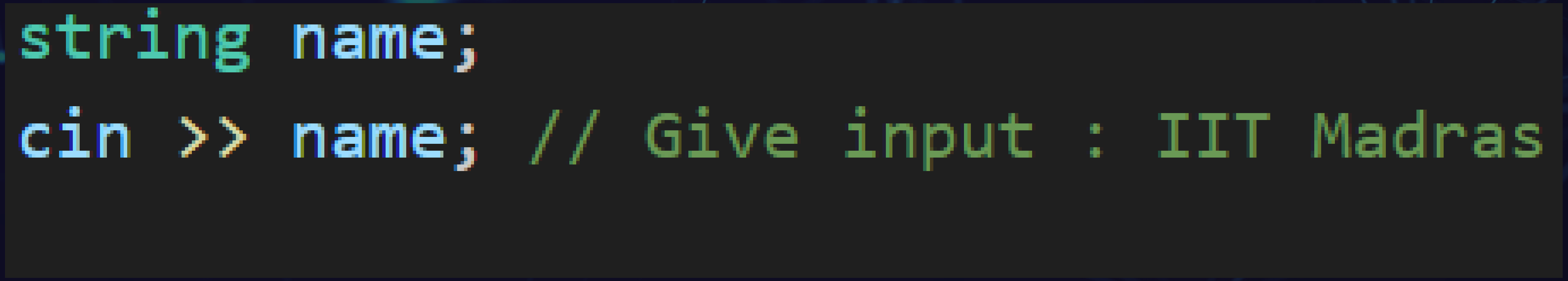
first.append(second) ?!

STRINGS

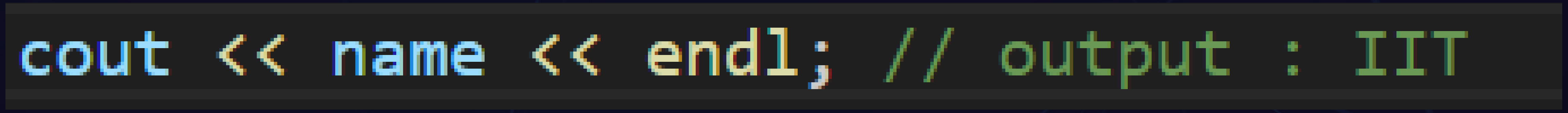
(1D ARRAY OF CHARACTERS)



```cpp
string name = "IIT Madras";

cout << name.size() << endl; // output : 10.
cout << name.length() << endl; // output : 10.
```



```cpp
string name;
cin >> name; // Give input : IIT Madras
```



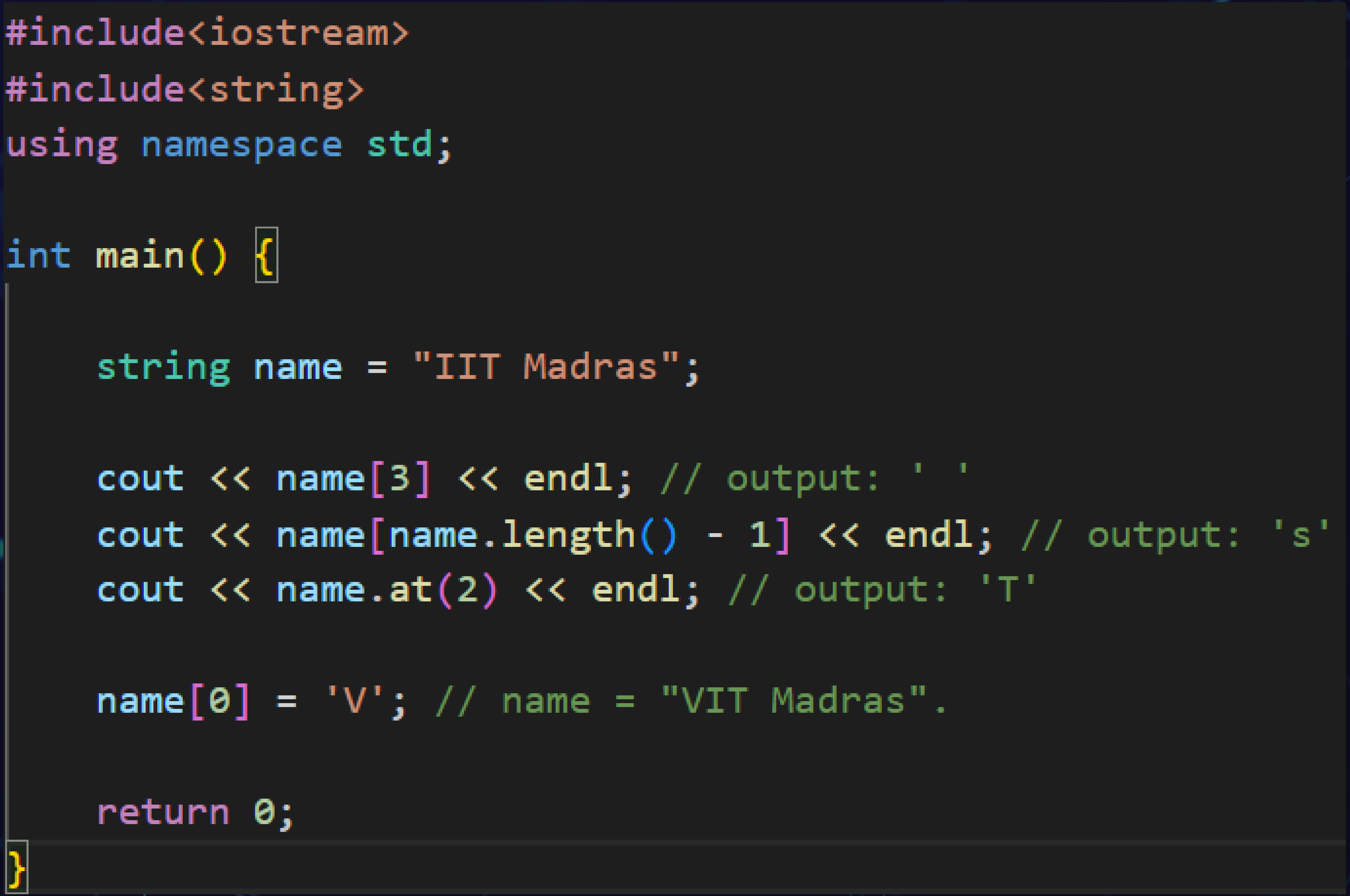
```cpp
cout << name << endl; // output : IIT
```

RIY

How to take the full line in input string ?!

STRINGS

(1D ARRAY OF CHARACTERS)



```
#include<iostream>
#include<string>
using namespace std;

int main() {

    string name = "IIT Madras";

    cout << name[3] << endl; // output: ' '
    cout << name[name.length() - 1] << endl; // output: 's'
    cout << name.at(2) << endl; // output: 'T'

    name[0] = 'V'; // name = "VIT Madras".

    return 0;
}
```

“IMAGINE HE HAS TO CALCULATE THE CLASS AVERAGE FOR DIFFERENT EXAMS?”

FUNCTIONS

- A block of code which runs when it is called.
- Important for reusing code.

Library functions : Included in header files.

sqrt(), abs() etc.

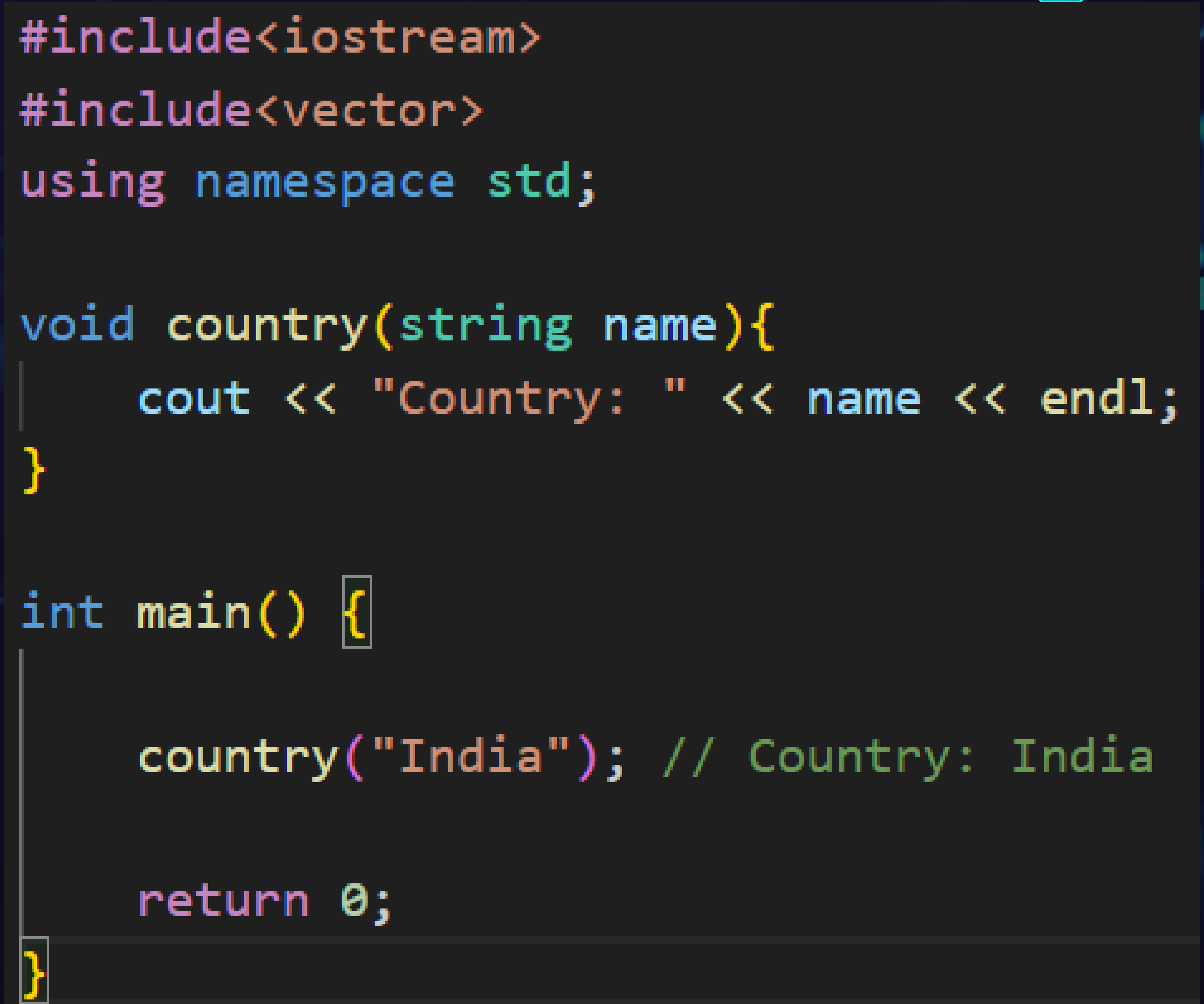
User-defined functions

Syntax ==>

```
return_datatype function_name(datatype1 a, datatype2 b)
```

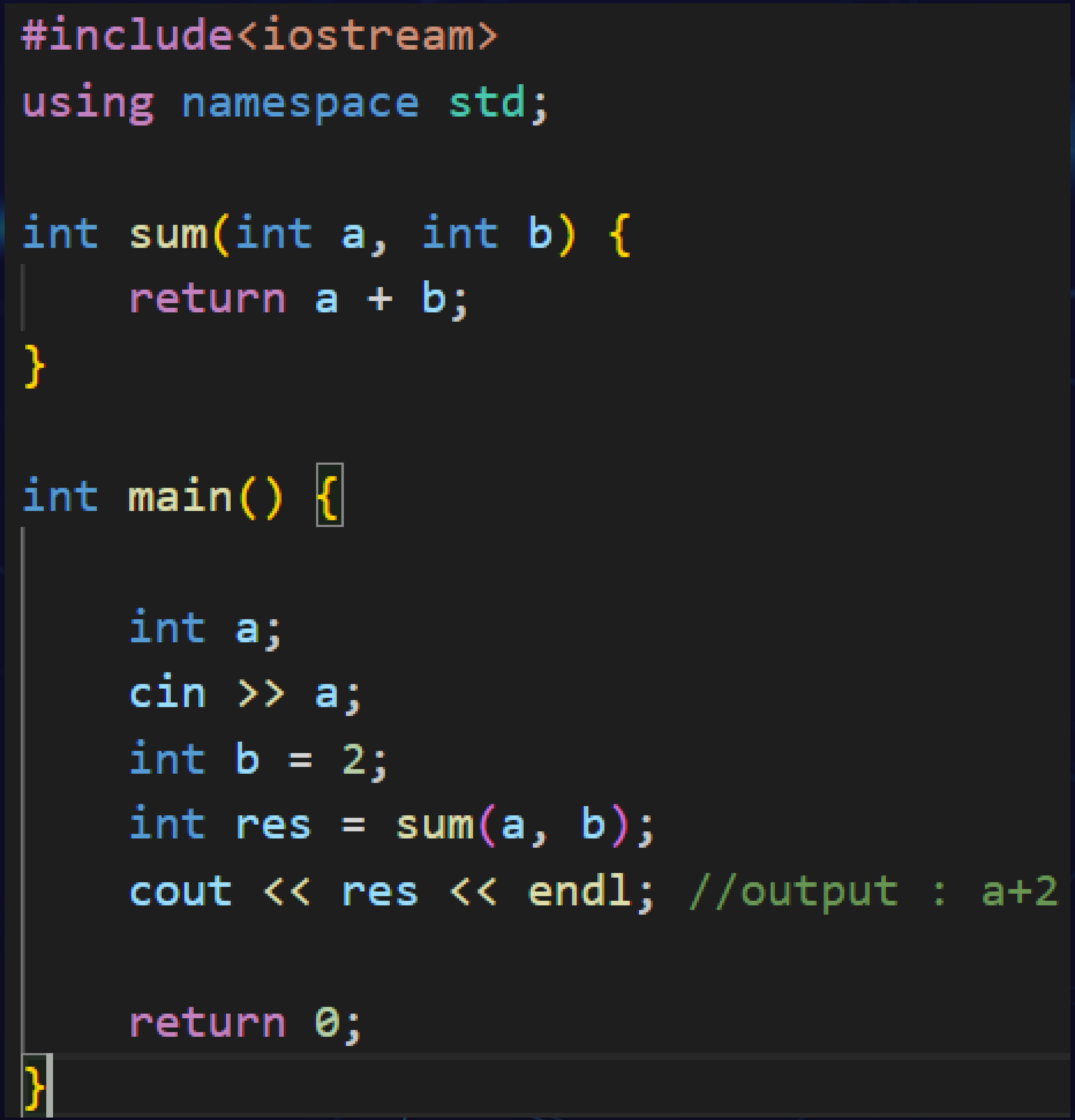
- int main() : used to execute code.

FUNCTIONS



```cpp
#include<iostream>
#include<vector>
using namespace std;

void country(string name){
    cout << "Country: " << name << endl;
}

int main() {

    country("India"); // Country: India

    return 0;
}
```

FUNCTIONS

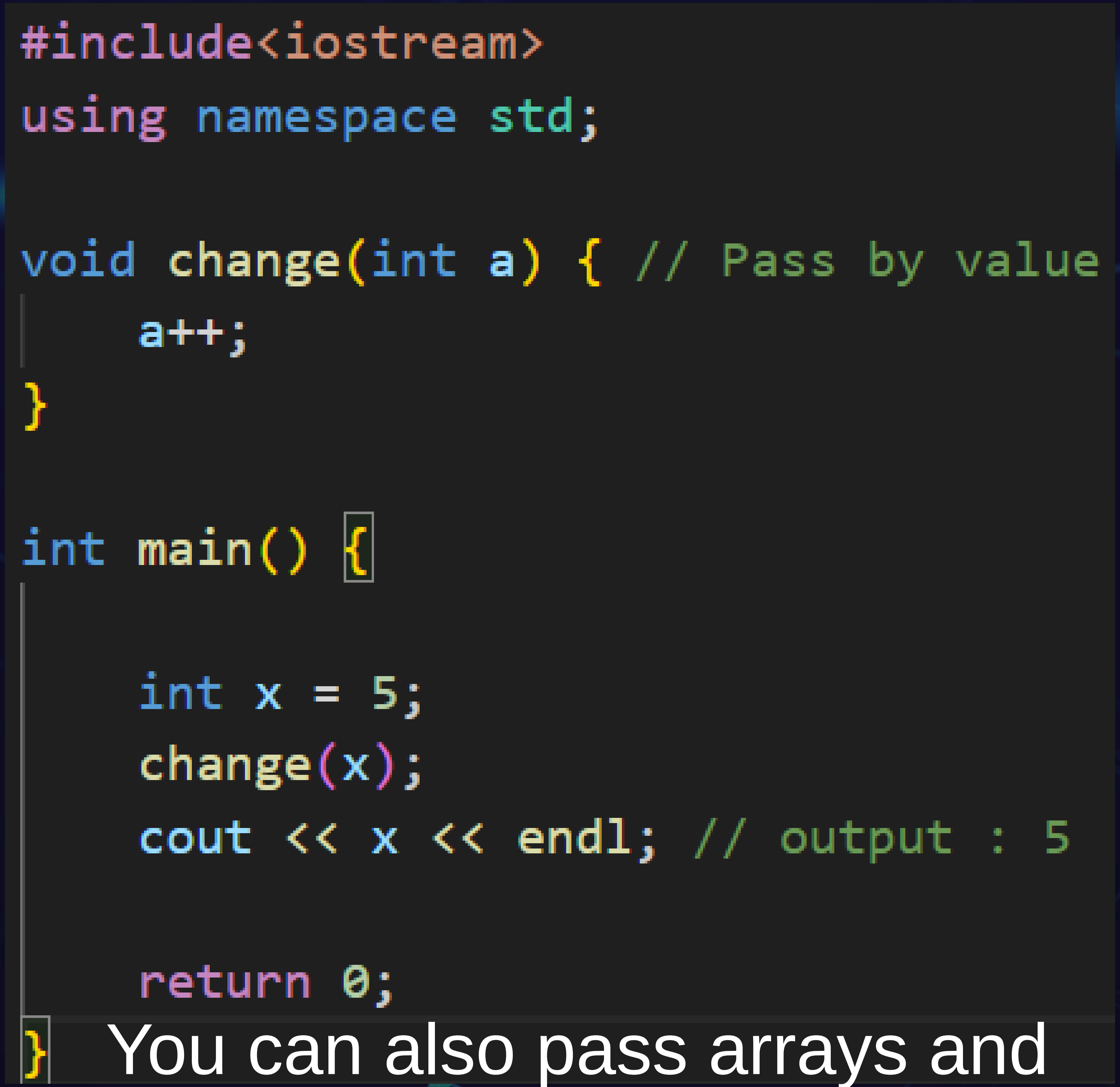


```
#include<iostream>
using namespace std;

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

int main() {

    int a;
    cin >> a;
    int b = 2;
    int res = sum(a, b);
    cout << res << endl; //output : a+2

    return 0;
}
```

DIY

Write a function which takes integer input and returns the factorial !

FUNCTIONS



```cpp
#include<iostream>
using namespace std;

void change(int a) { // Pass by value
    a++;
}

int main() {

    int x = 5;
    change(x);
    cout << x << endl; // output : 5

    return 0;
}
```

You can also pass arrays and vectors into the function.

<u>Pass by reference</u>

```cpp
void change(int &a) {
    a++;
}
```

RIY

Pass by pointers !?

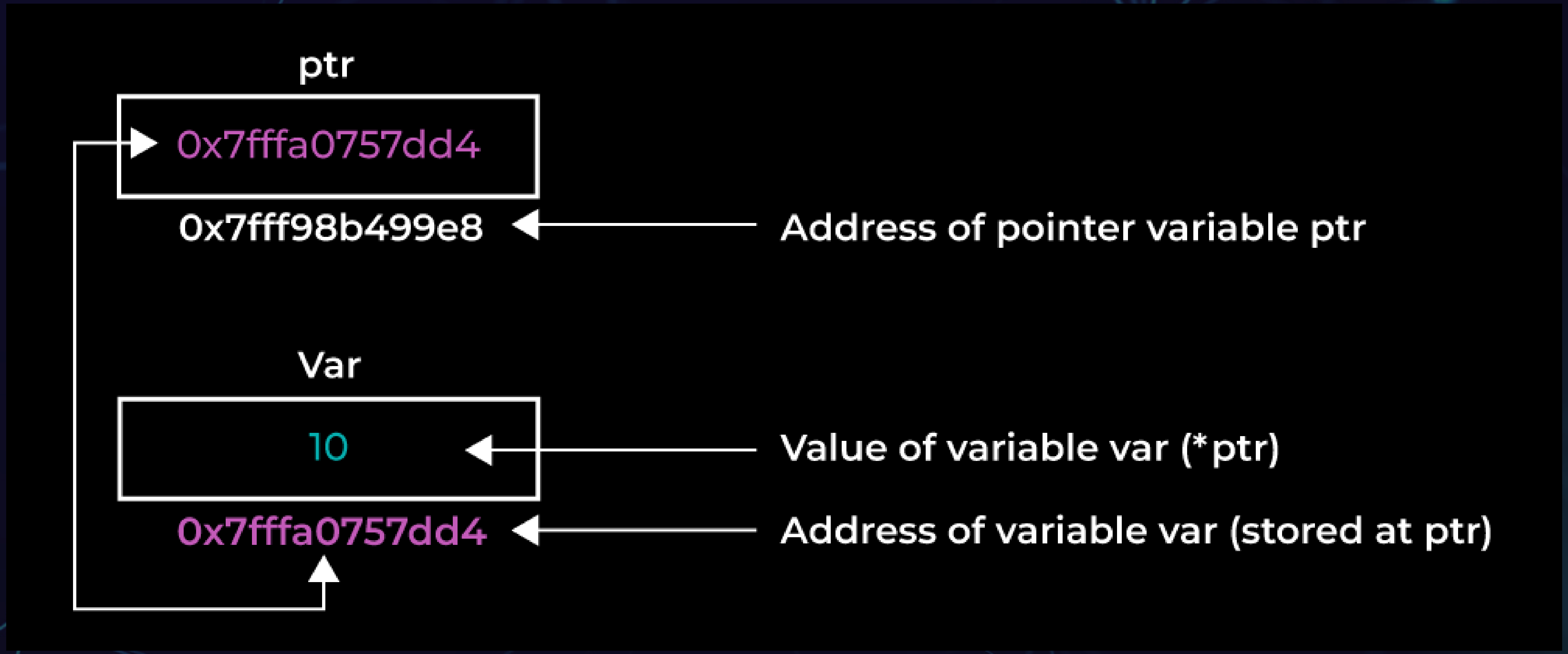
Agenda

List of topics we will cover today:

- Pointers
- References
- Classes
- Access specifiers
- Structs

Introduction to Pointers:

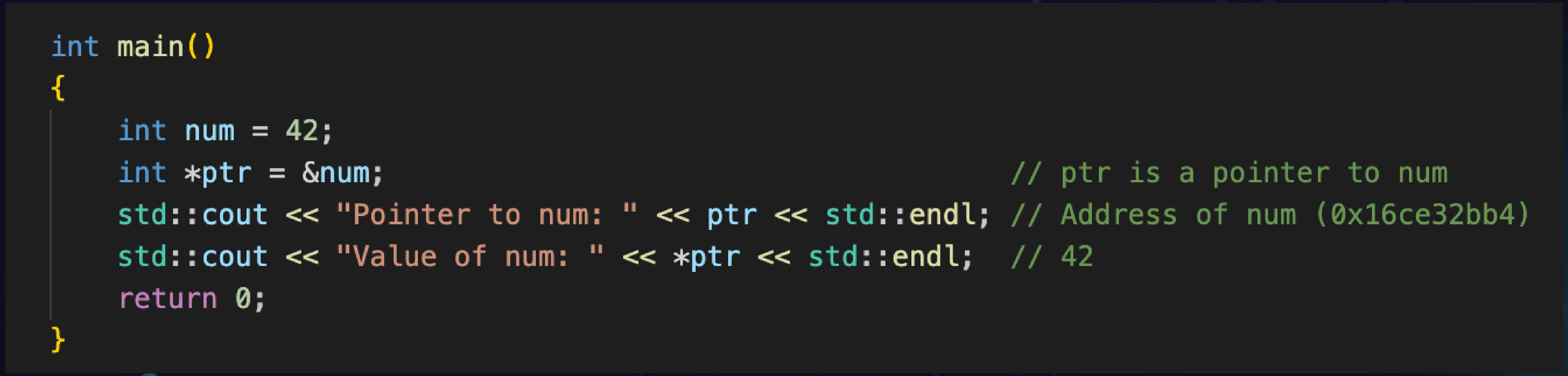
- Memory locations have addresses.
- A pointer is just an integer (long long) variable that stores the address of another variable.


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (stored at ptr)

Introduction to Pointers:

- & is the address-of operator. It returns the memory address of the operand.

- * is the indirection operator. It is used to “dereference” a pointer, i.e., return the value stored at the memory address it is pointing to.



```cpp
int main()
{
    int num = 42;
    int *ptr = &num;                                     // ptr is a pointer to num
    std::cout << "Pointer to num: " << ptr << std::endl; // Address of num (0x16ce32bb4)
    std::cout << "Value of num: " << *ptr << std::endl;  // 42
    return 0;
}
```

Question

- What is the size of a pointer? Is it dependent on the type of the variable it points to?

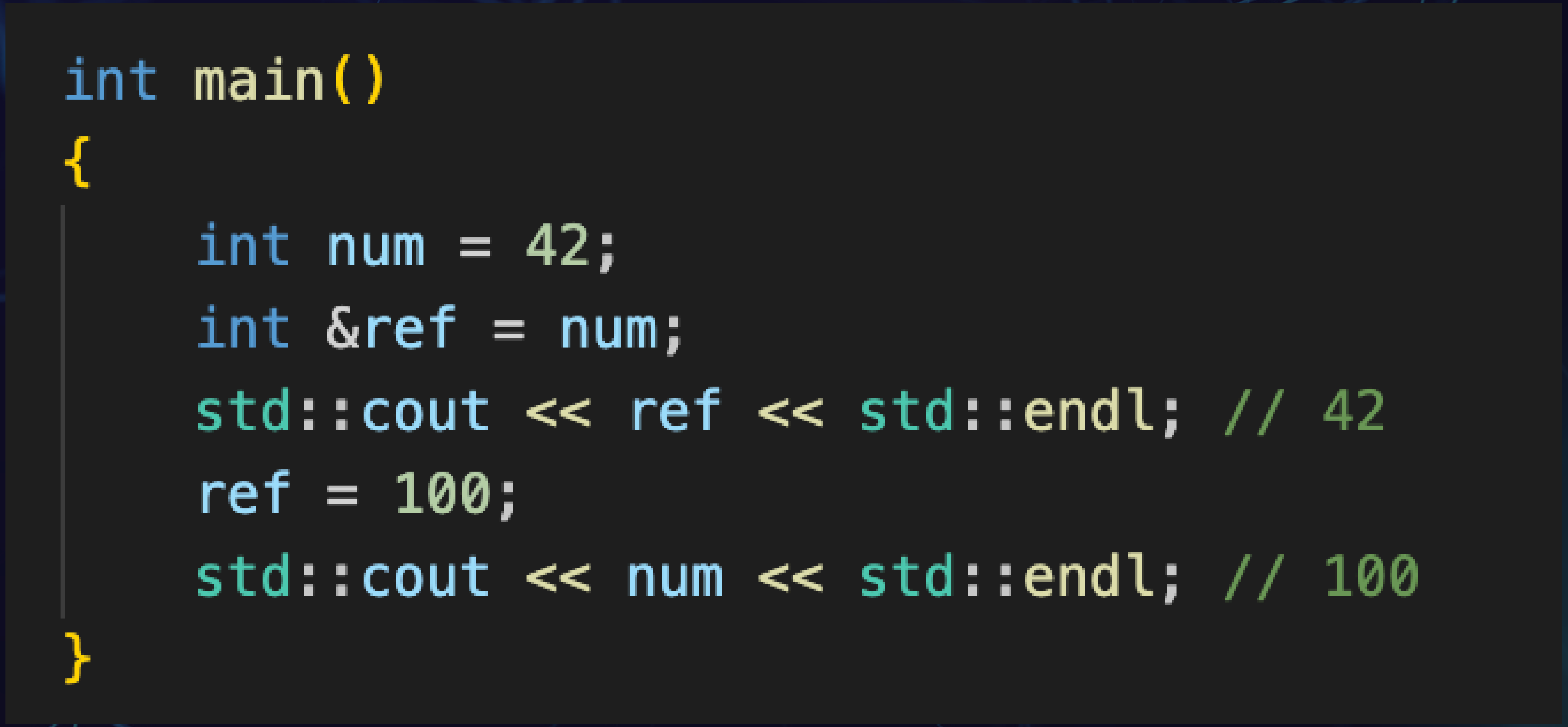
Question

- What is the size of a pointer? Is it dependent on the type of the variable it points to?

- “If a pointers is just an integer (or long long), why do we need to specify datatype of the variable while declaring a pointer for other datatypes like string, bool, etc.”

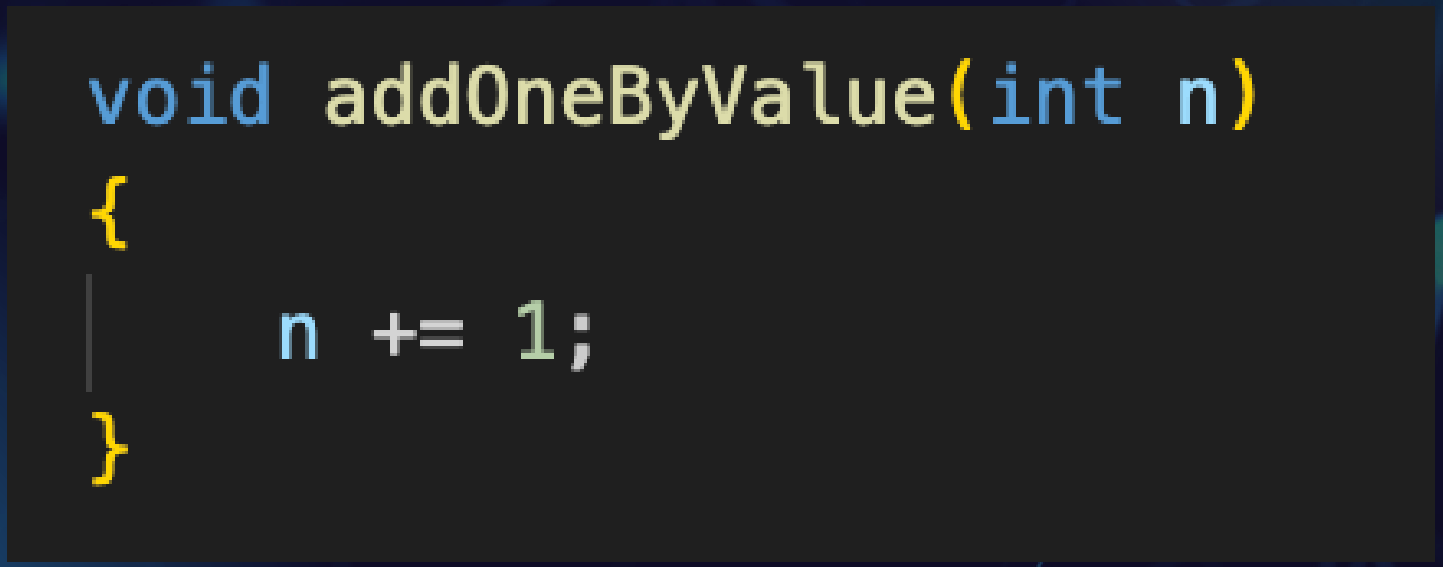
References

- References are an alias for an existing variable, providing an alternative name for it and allowing you to work with the original data directly.

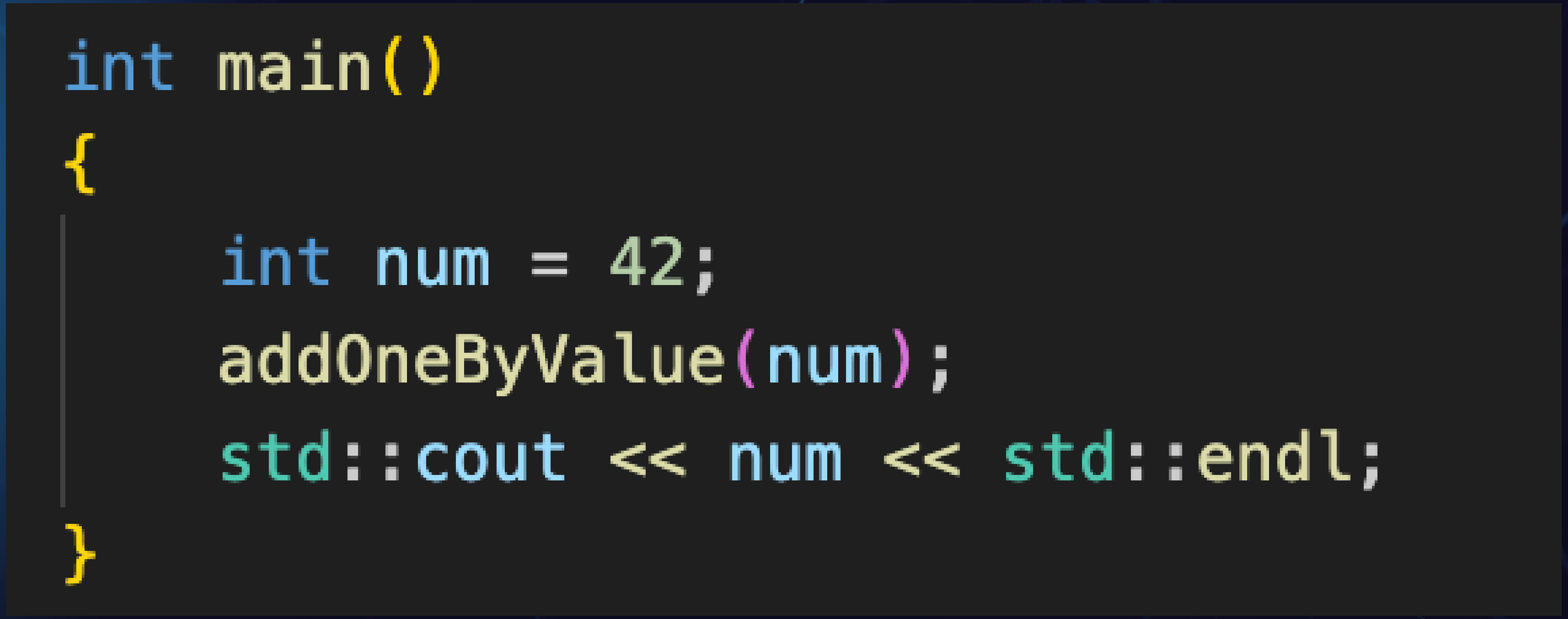


```cpp
int main()
{
    int num = 42;
    int &ref = num;
    std::cout << ref << std::endl; // 42
    ref = 100;
    std::cout << num << std::endl; // 100
}
```

Pass By Value



```
void addOneByValue(int n)
{
    n += 1;
}
```



```
int main()
{
    int num = 42;
    addOneByValue(num);
    std::cout << num << std::endl;
}
```

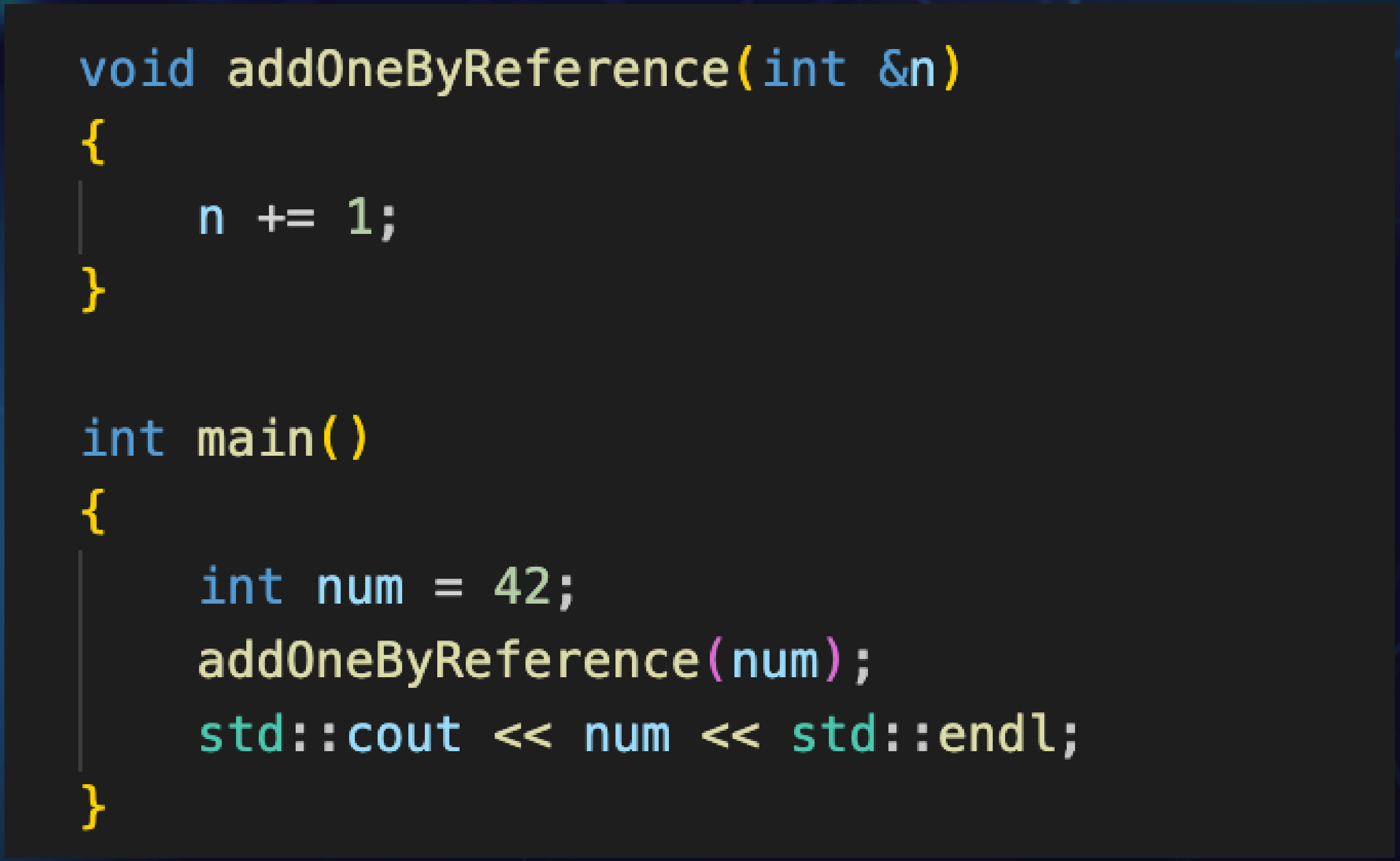
Predict the output!

Pass By Reference

```cpp
void addOneByReference(int &n)
{
    n += 1;
}

int main()
{
    int num = 42;
    addOneByReference(num);
    std::cout << num << std::endl;
}
```

Predict the output!

Pass By Reference



```
void addOneByReference(int &n)
{
    n += 1;
}

int main()
{
    int num = 42;
    addOneByReference(num);
    std::cout << num << std::endl;
}
```

Can you get the same behaviour using pointers??

References vs Pointers

- References must be initialized, and cannot be changed to refer elsewhere. There are no “null” references.
- Pointers can be reassigned. There are explicit * and & operators. nullptr can be used.

References are just safer, cleaner aliases.

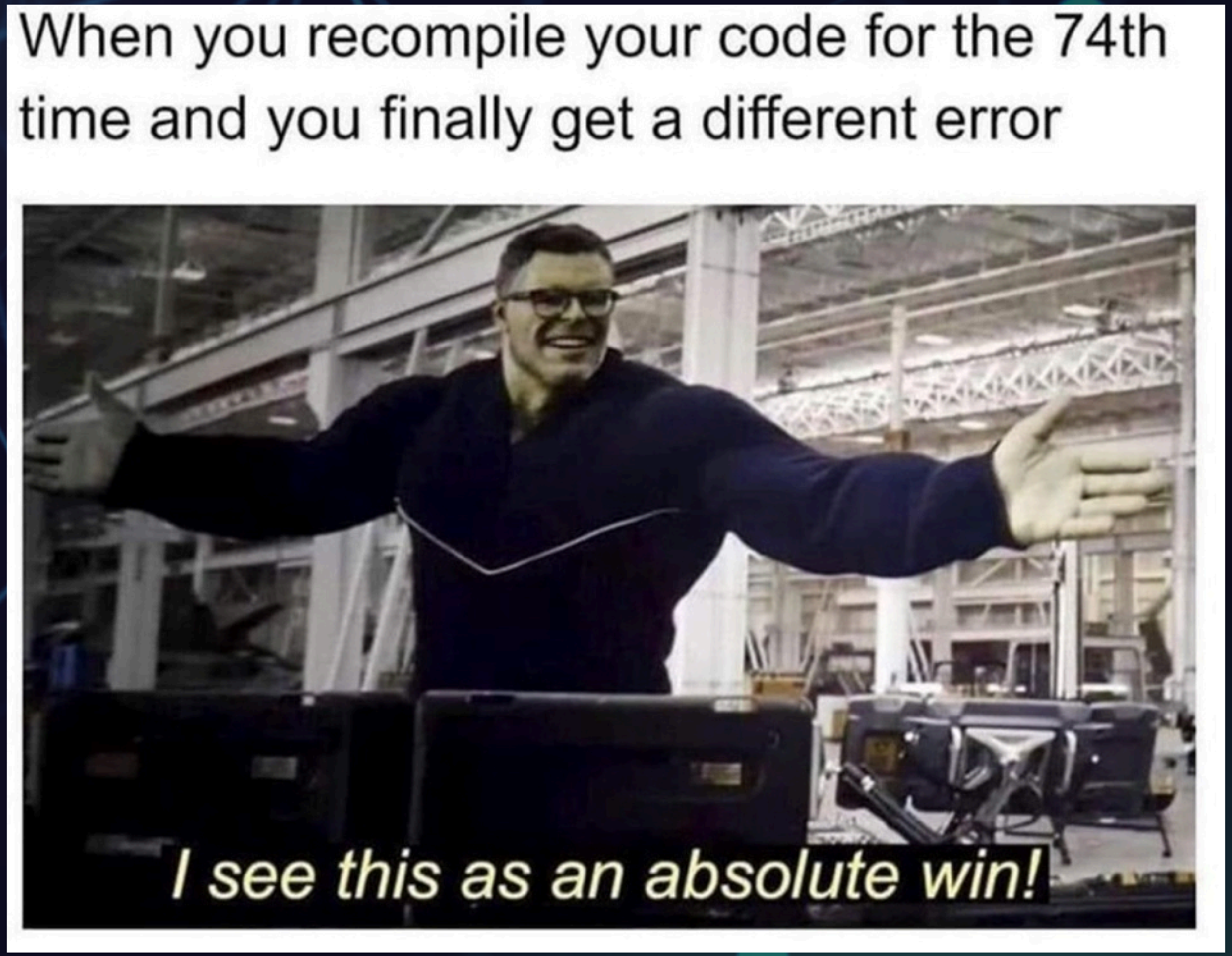
If you need re-pointing or nullability, use pointers.

BUT WAIT.....

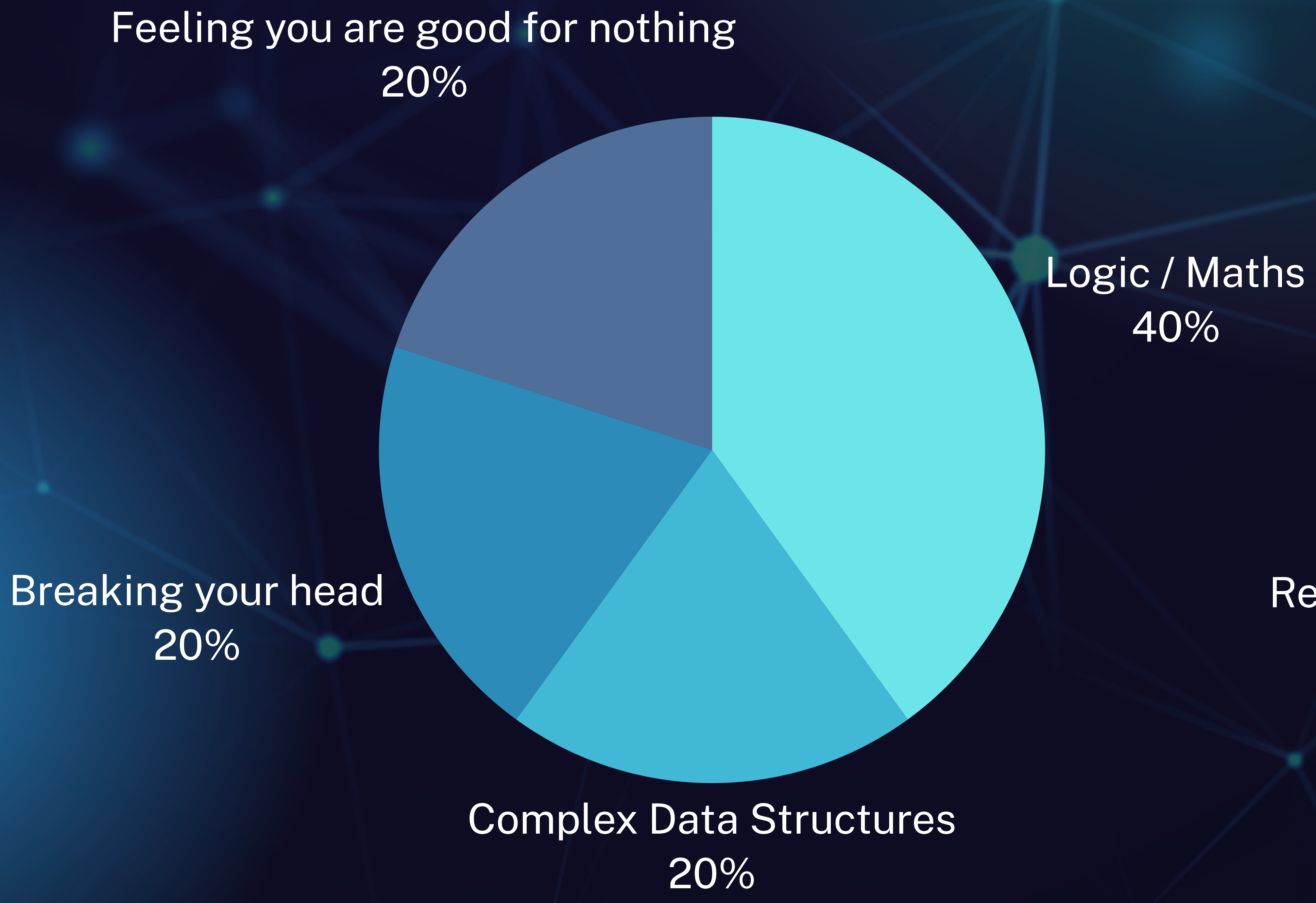
WHAT IS SOFTWARE DEVELOPMENT ??

What is software dev?

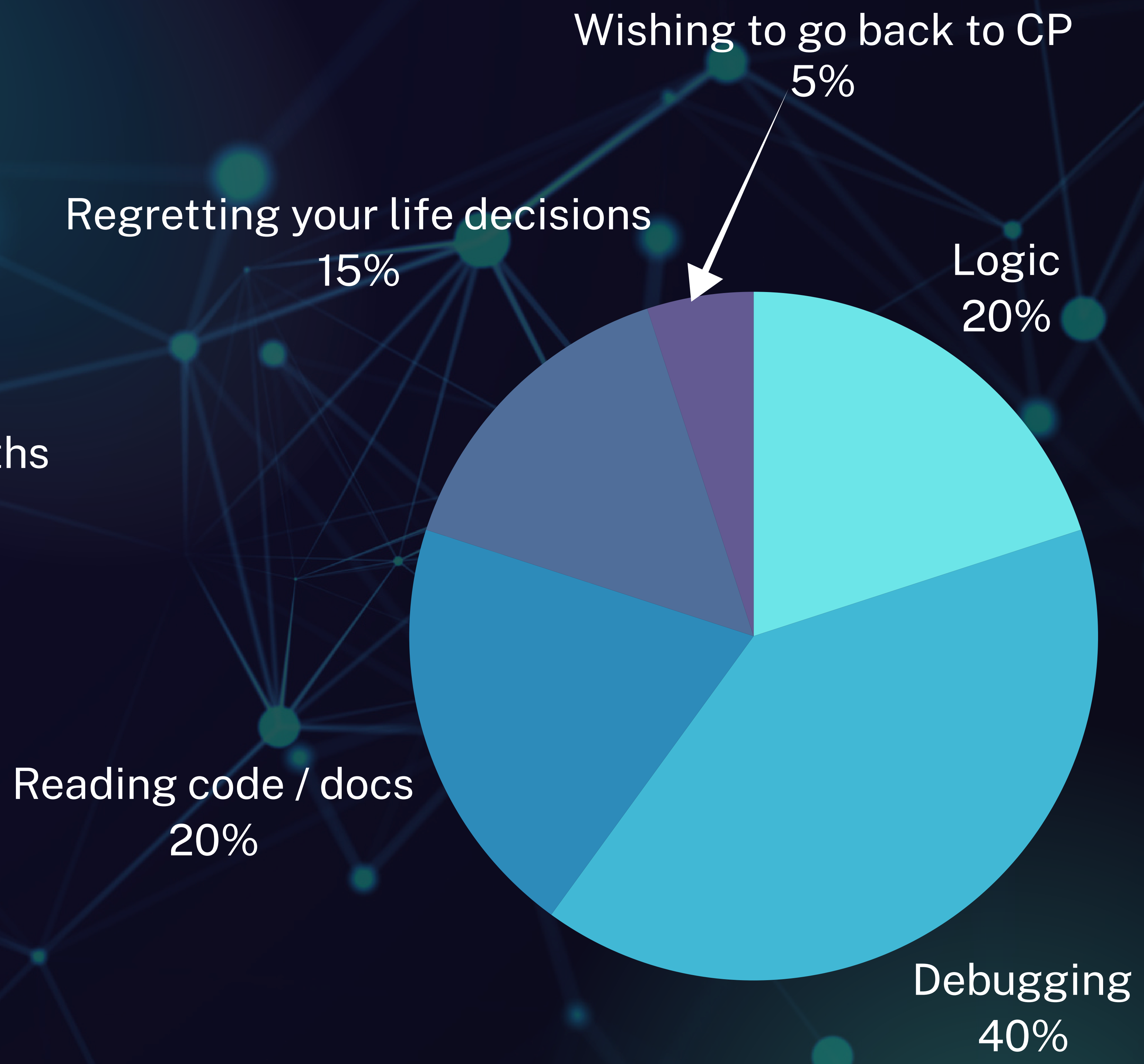
- Software Development is writing code that does what you want to do except the fact that you are not just sitting and solving CF problems
- More than just coding - documentation and designing also matters
- Reading code is crucially important and debugging is everything in SD.


When you recompile your code for the 74th time and you finally get a different error
I see this as an absolute win!

And yeah this too ...



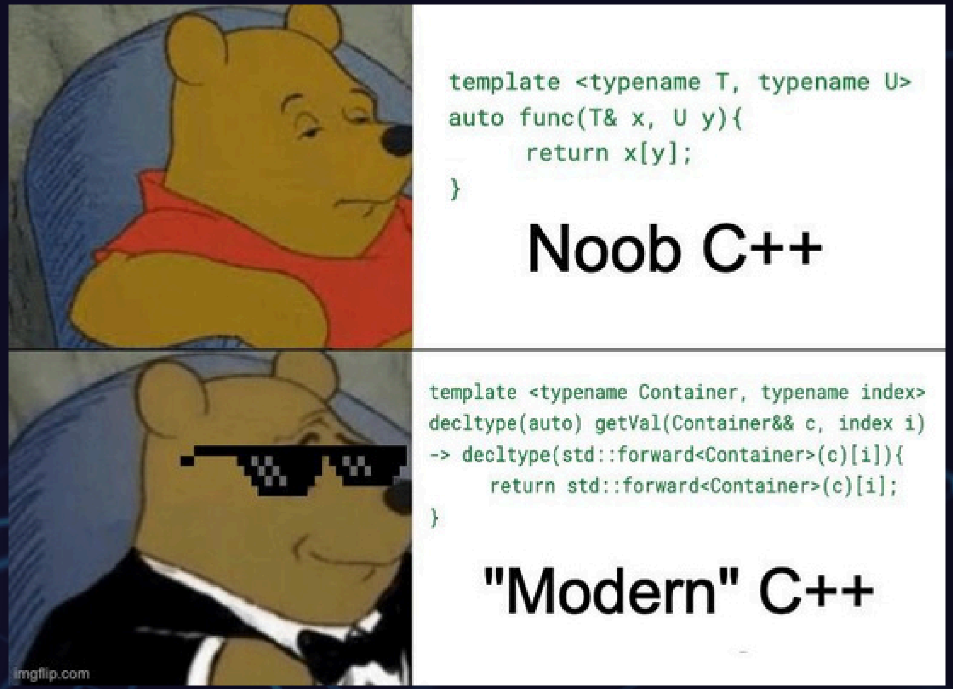
Competitive Programming (CP)



Software Development (SD)

WE FIND THE BUG
WE FIX THE BUG
NOW WE HAVE TWO BUGS
NOW WE HAVE THREE BUGS
imgflip.com

Dev vs CP



```
template <typename T, typename U>
auto func(T& x, U y){
      return x[y];
}
```

```
template <typename Container, typename index>
decltype(auto) getVal(Container&& c, index i)
-> decltype(std::forward<Container>(c)[i]){
    return std::forward<Container>(c)[i];
}
```

Aspect	**CP**	**Dev**
Goal	Solve problem quickly	Build robust, long-term systems
Timeframe	Minutes to hours	Months to years
Focus	Algorithms, speed	Design, readability, maintainability
Teamwork	Solo (generally)	Collaborative
Testing	Get AC and pack	Extensive unit & integration testing
Code Style	Short, optimized	Clear, well-documented

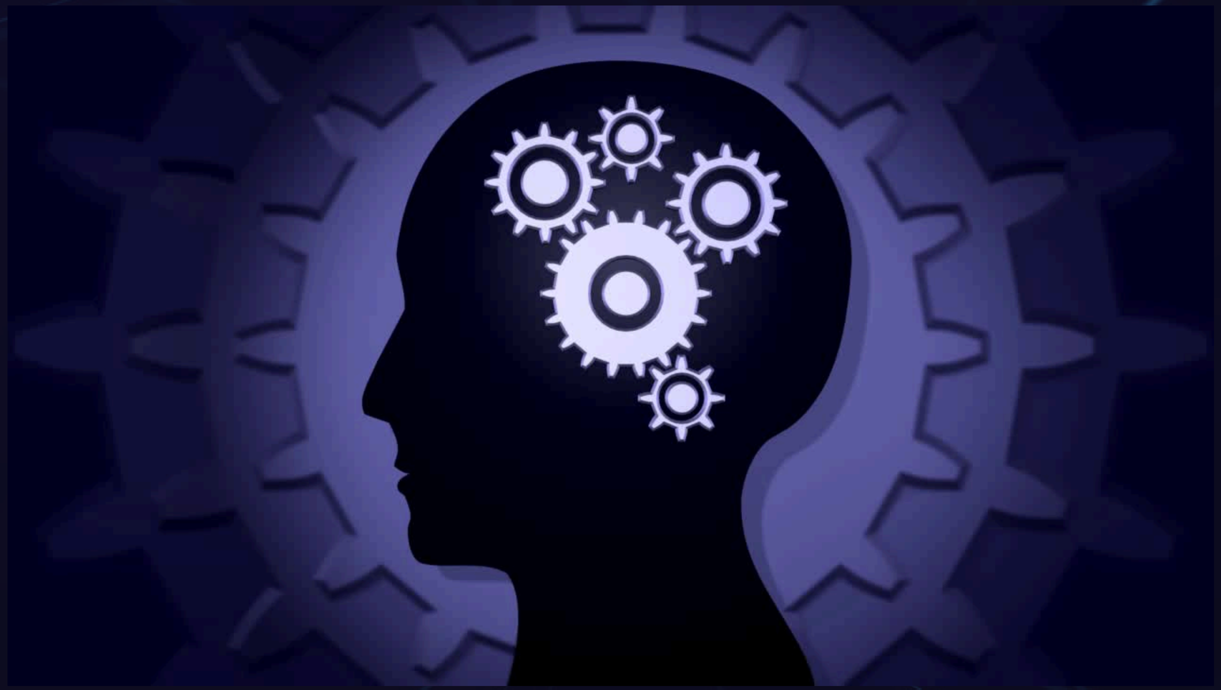
What is a Class?

- A class is a user-defined type that has variables and functions as its members.
- Basically a way of grouping some data and/or functionality together.
- A feature of C++, an object-oriented language.

Members of Class

- Data Members: Store attributes/state of an object (What an object has)

- Member Functions: Define behavior of the class. They are also called methods. (What an object does)

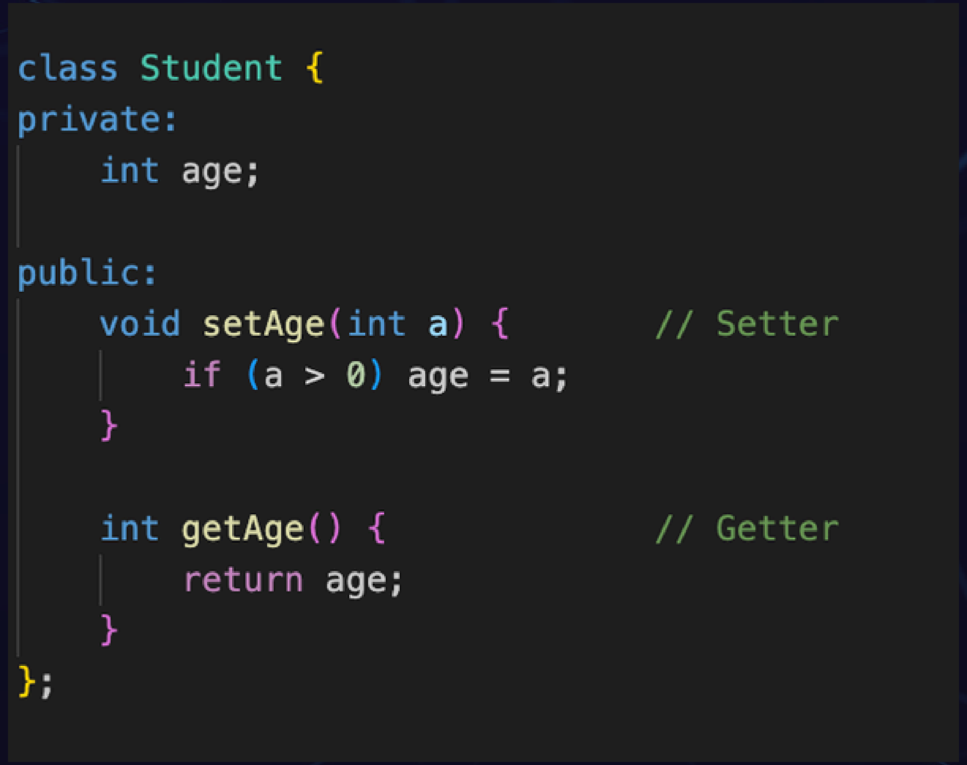


Access Specifiers

- **Public –** Accessible from anywhere in the program
- **Private** – Accessible only from the class it is defined in. Cannot access from derived classes or any other part of the program.
- **Protected** – Accessible from the class it is defined in and it’s derived classes. Cannot access from any other part of the program.

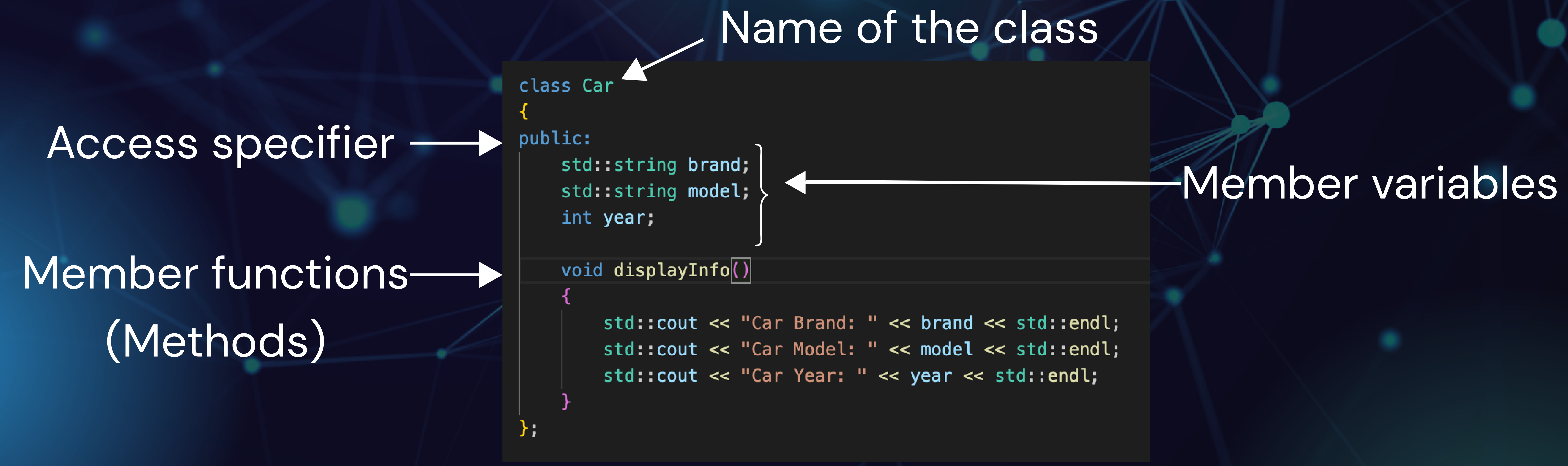
Setters and getters

- Setters: Function that updates the value of a private data member.
- Getters: Function that returns the value of a private data member.
- These functions act like gates to access and update private variables.



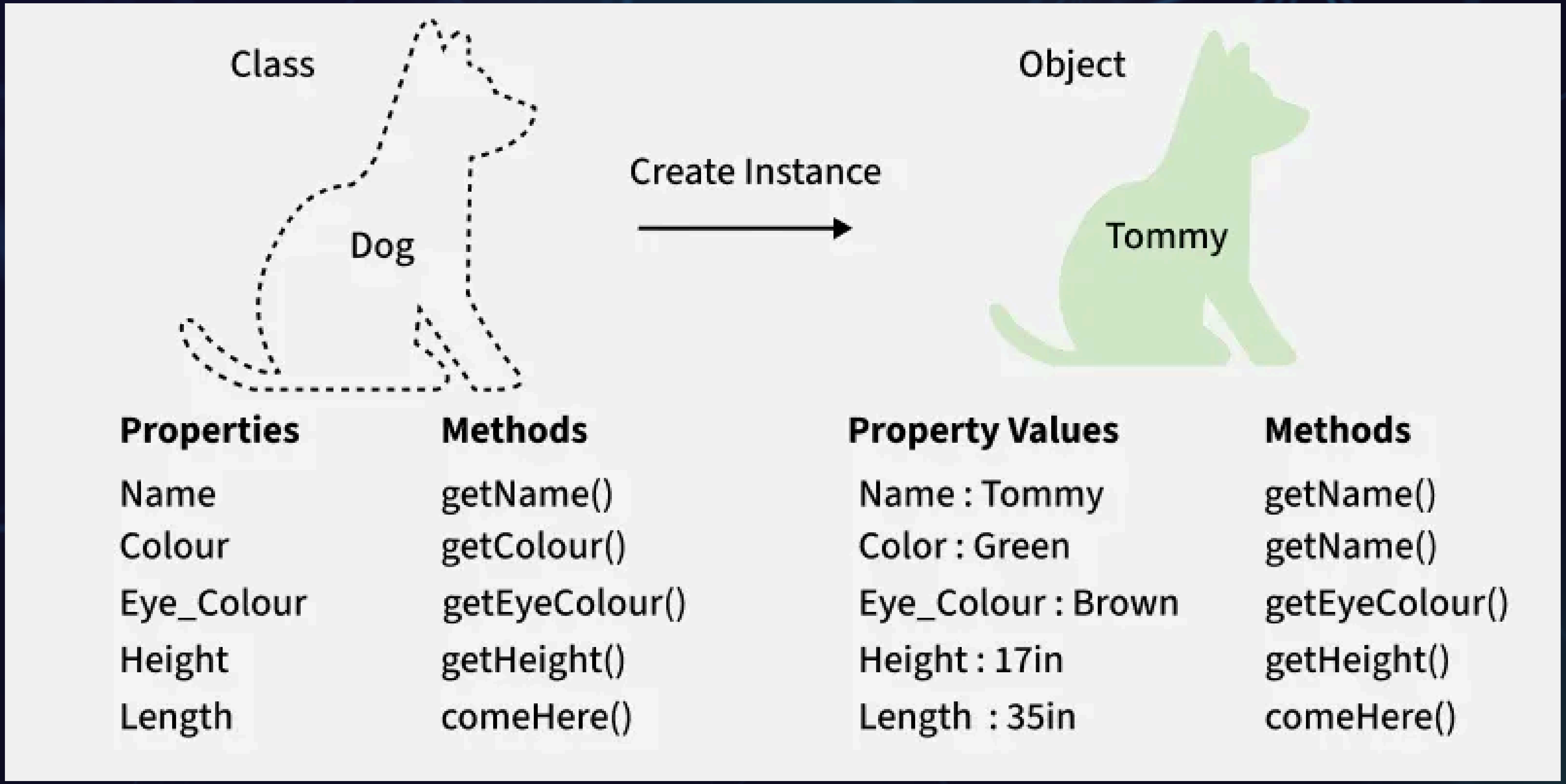
```
class Student {
private:
    int age;

public:
    void setAge(int a) {       // Setter
        if (a > 0) age = a;
    }

    int getAge() {             // Getter
        return age;
    }
};
```

Syntax



```cpp
class Car
{
public:
    std::string brand;
    std::string model;
    int year;

    void displayInfo()
    {
        std::cout << "Car Brand: " << brand << std::endl;
        std::cout << "Car Model: " << model << std::endl;
        std::cout << "Car Year: " << year << std::endl;
    }
};
```

Example



Properties	Methods	Property Values	Methods
Name	getName()	Name : Tommy	getName()
Colour	getColour()	Color : Green	getName()
Eye_Colour	getEyeColour()	Eye_Colour : Brown	getEyeColour()
Height	getHeight()	Height : 17in	getHeight()
Length	comeHere()	Length : 35in	comeHere()

Source: GFG

Structs

- Everything is same as Classes, the only difference is that, in Structs, all members are public by default, whereas in Classes, all members are private by default.

- But it can contain both private and public members by explicitly defining.

```
template<class T>
concept isBar = requires(T obj, const Span& sp, Bar&& bs) {
    obj.bar(sp, bs);
};
```

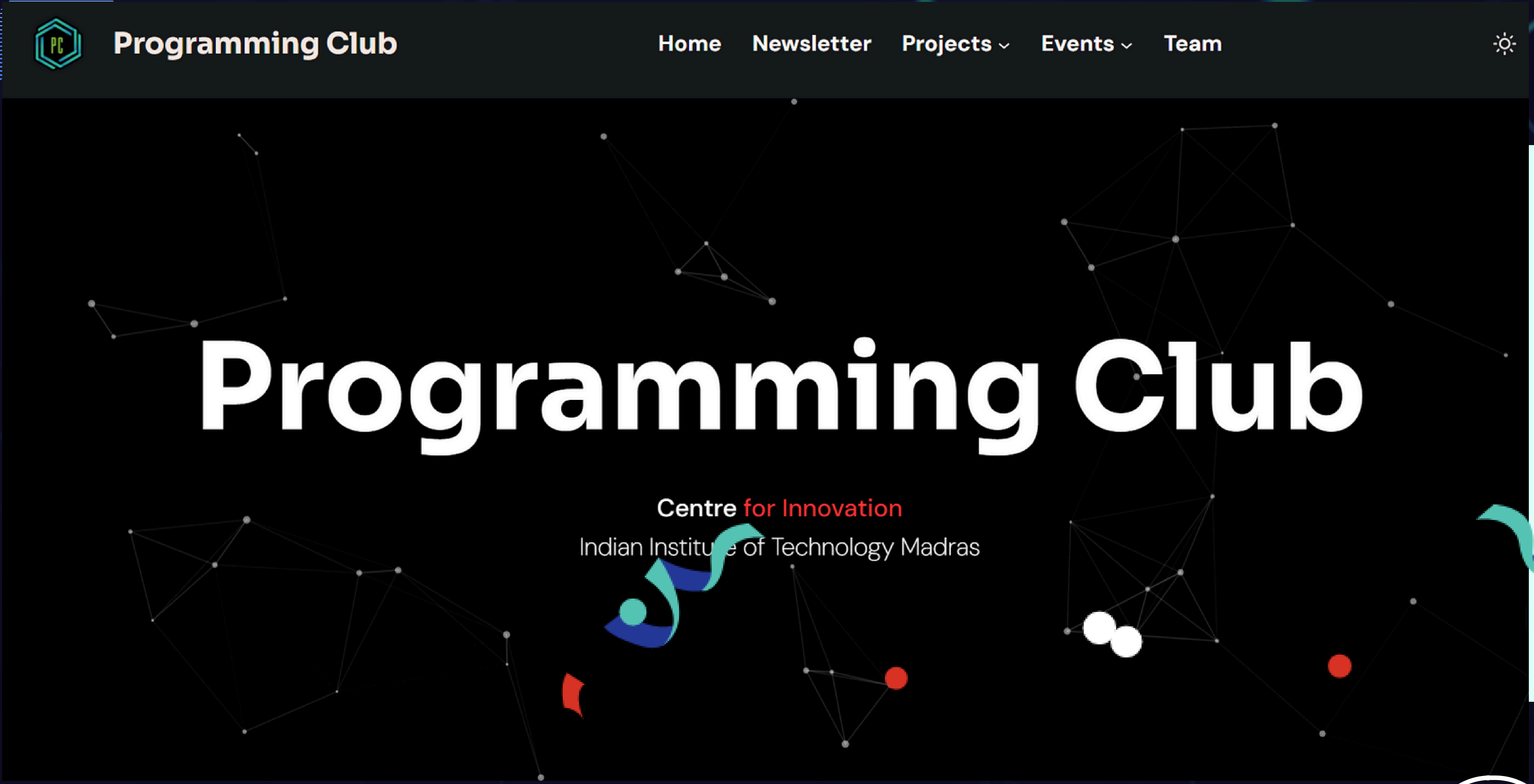
MY CONCEPTS LOOK LIKE THIS

```
template<class T>
    requires isBar<T>
void foo(T obj, const Span& sp, Bar&& bs) {
    obj.bar(sp, bs);
}
```

SO HER CODE CAN LOOK LIKE THIS
They're the same picture.

One more thing..

OUR WEBSITE



Programming Club

Centre for Innovation

Indian Institute of Technology Madras

https://progclubiitm.com