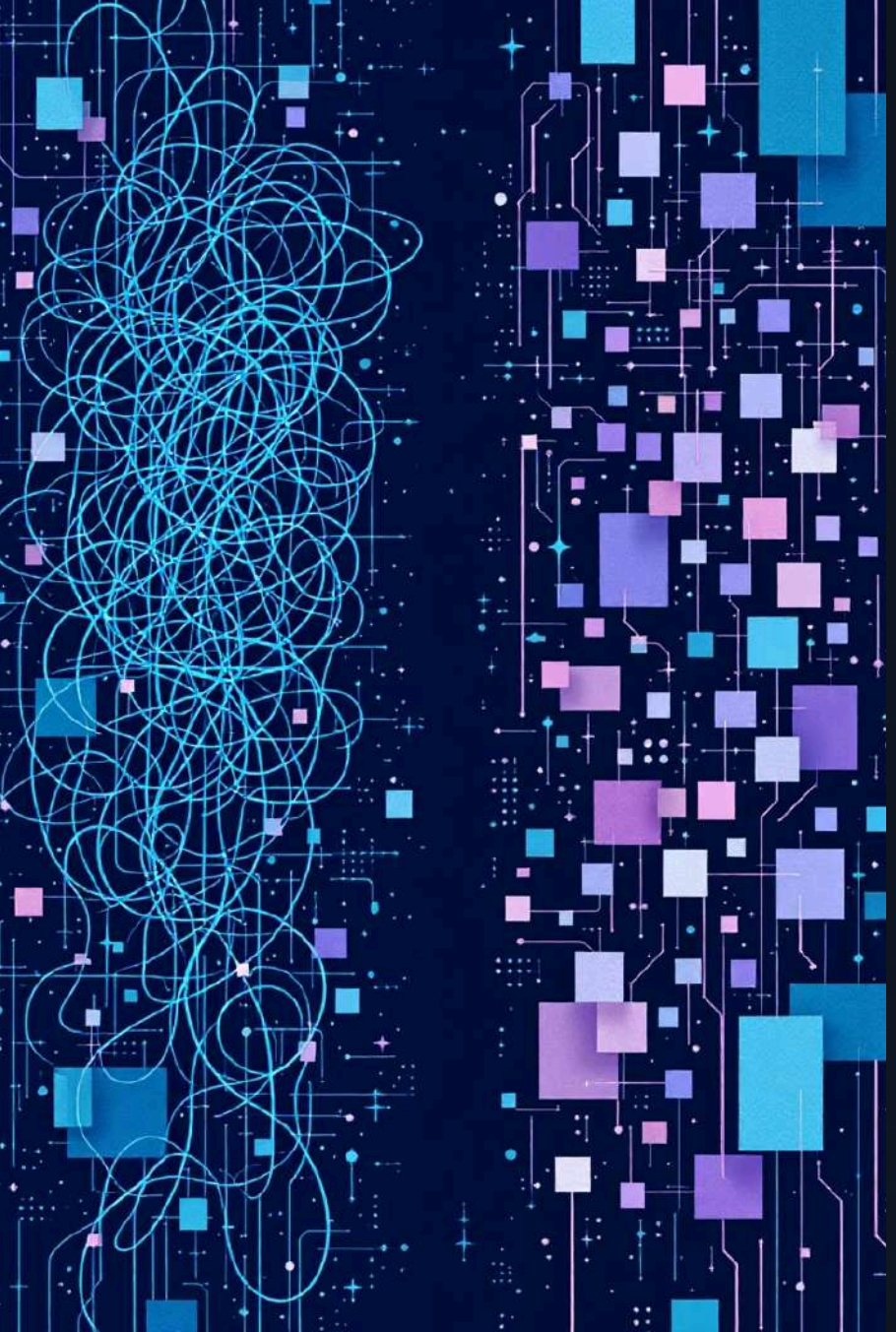
Functions and Modularity in C++

Exploring the principles of creating structured and reusable code



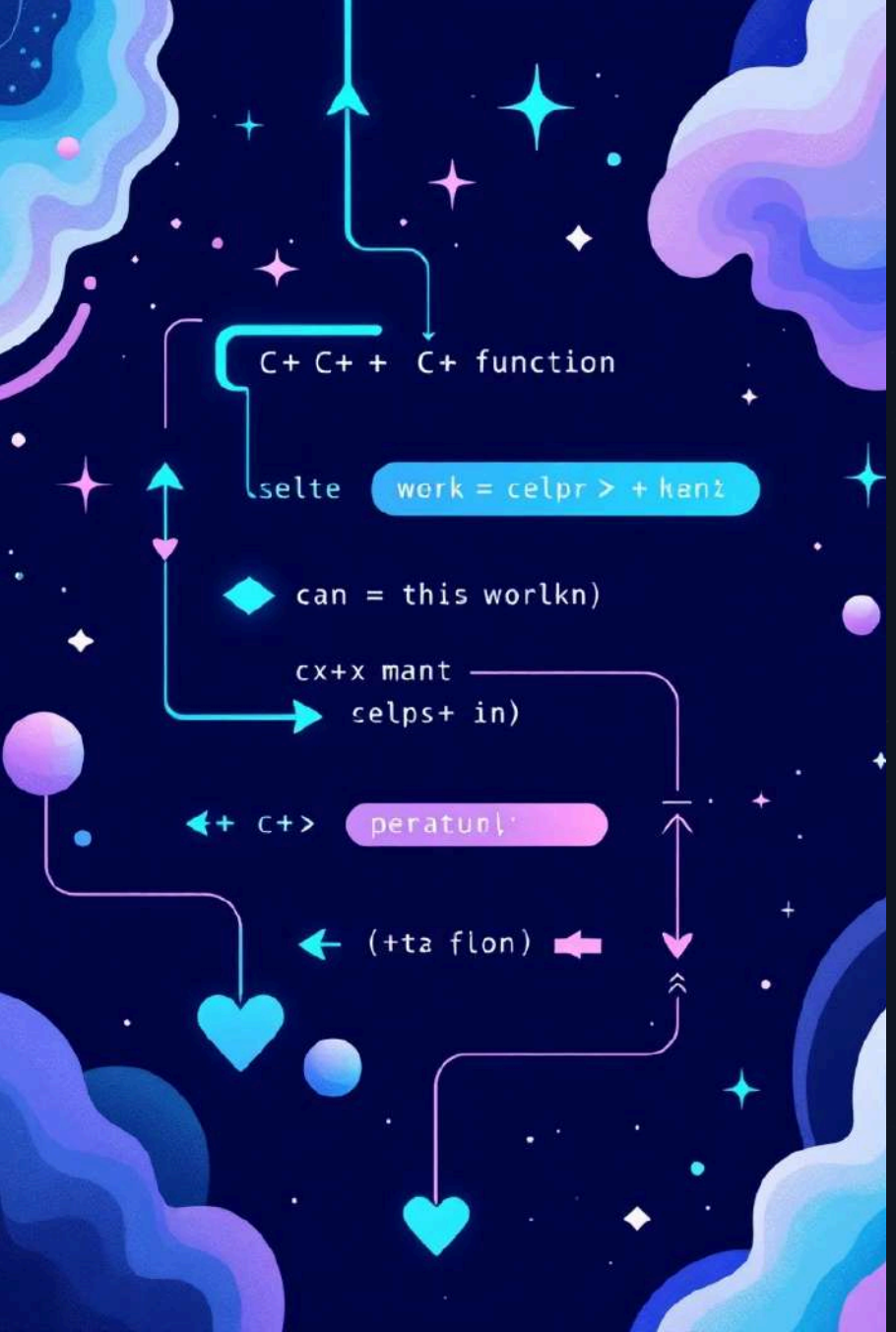
Why Functions?

Problem

Large programs are difficult to read, maintain, and debug. Code duplication leads to errors.

Solution

Functions and the principle of modularity help break down complex tasks into simple parts.



C++ Function Structure

```
return_type function_name(parameter_list) {
    // function body
    return value; // (if return_type != void)
}
```

01

Return Type

Indicates the data type the function returns (int, double, void, etc.)

02

Function Name

Unique identifier for calling the function

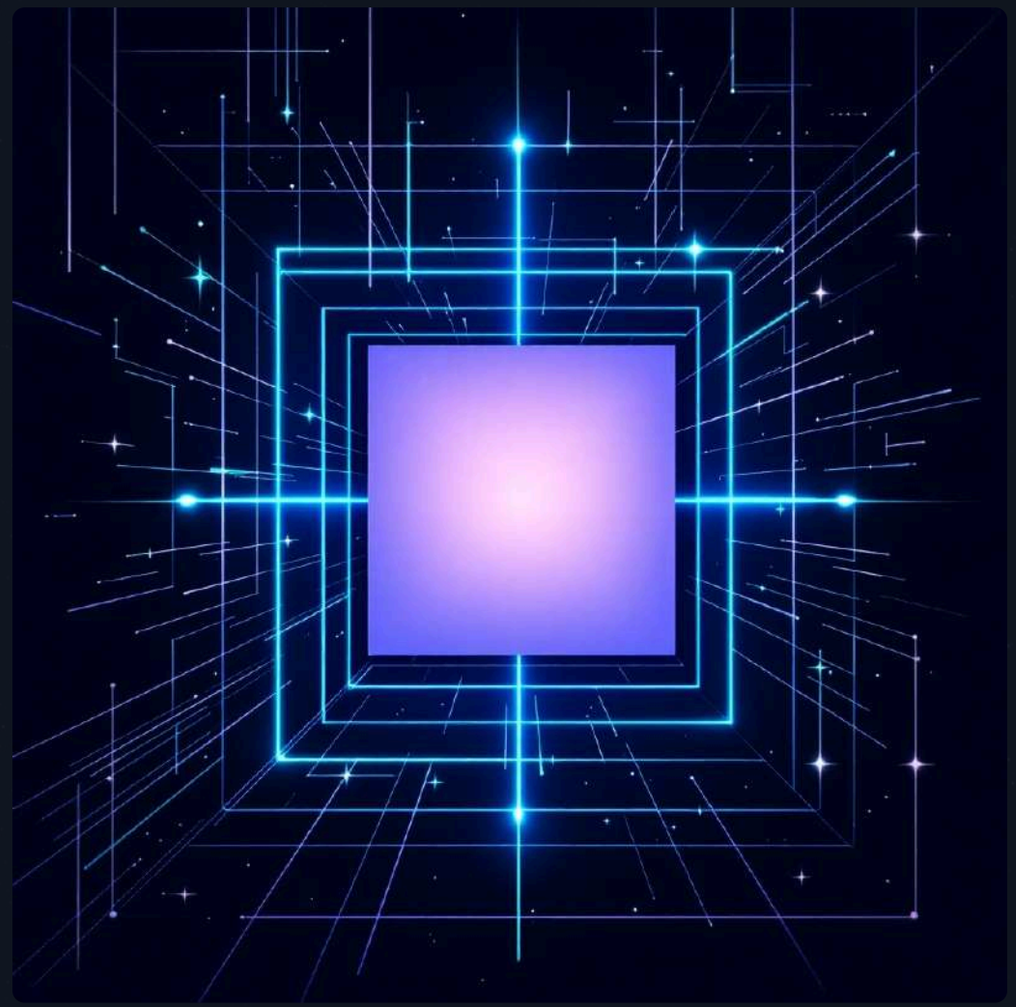
03

Parameter List

Input data that the function accepts to perform its operations

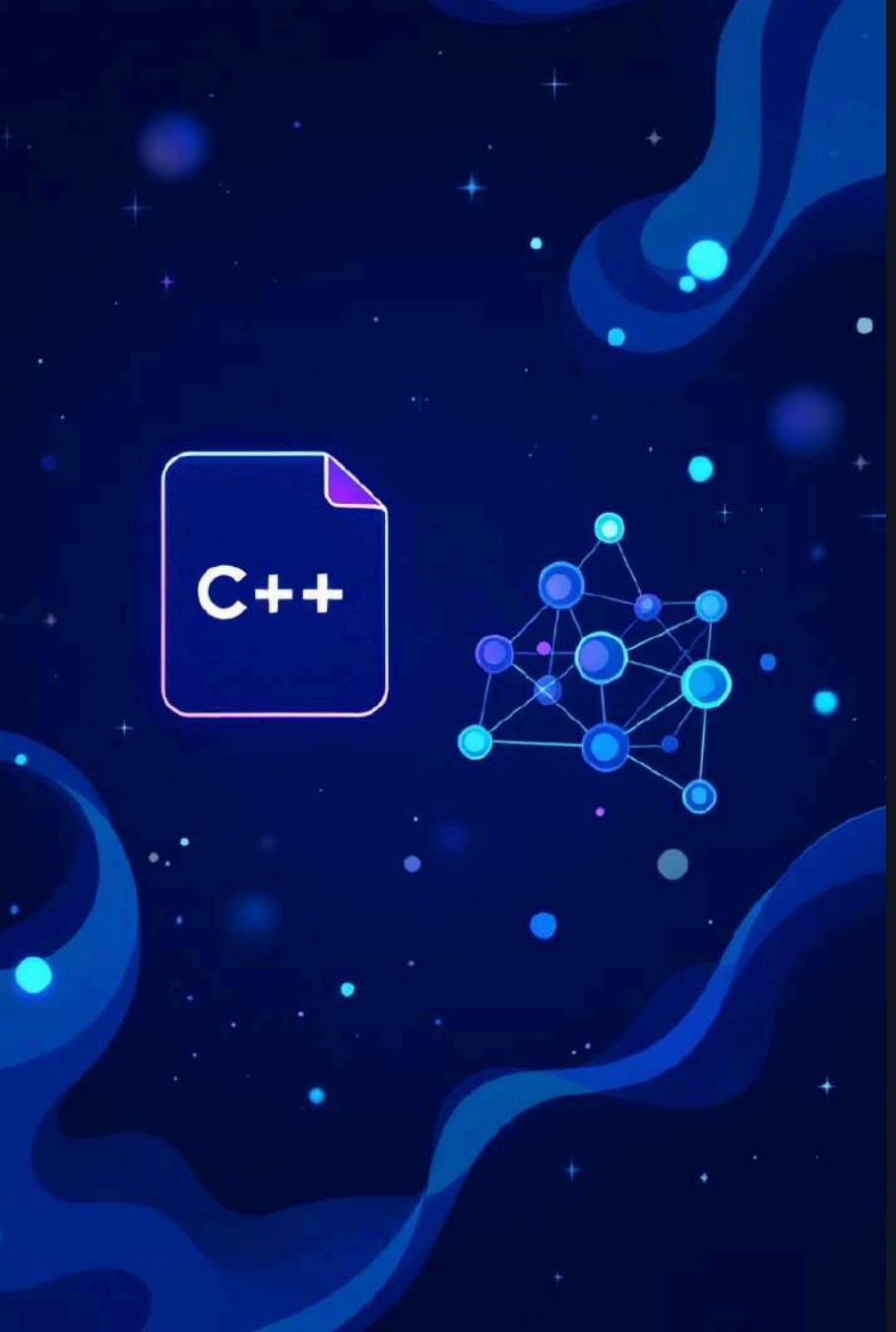
Function Example in Action

```
#include
using namespace std;

int square(int x) {
    return x * x;
}

int main() {
    cout << "Square of 5 = "
         << square(5) << endl;
    return 0;
}
```



Result: Square of 5 = 25

The `square` function takes an integer and returns its square.



Declaration vs Definition

Function Prototype (Declaration)

```
int sum(int a, int b); // declaration
```

Informs the compiler about the function's existence

Function Definition

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

Contains the implementation of the function

Parameter Passing Methods

1

By Value (copy)

```
void increment(int x) {
    x++; // only the copy is changed
}
```

2

By Reference (&)

```
void increment(int &x) {
    x++; // the original variable is changed
}
```

3

By Pointer (*)

```
void increment(int *x) {
 (*x)++; // working via address
}
```

Recursive Functions

A function can call itself to solve sub-problems

Base Case

Condition for stopping recursion

```
if (n <= 1) return 1;
```

Recursive Case

The function calls itself

```
return n * factorial(n - 1);
```

Example: Calculating the factorial of a number

Modular Code Organization

math_utils.h

Header file contains function declarations and include guards

math_utils.cpp

Source file contains function implementations

main.cpp

Main program uses functions by including the header

Key Principles

Functions – the basis of structuring

Break down complex tasks into simple functions

Choose the correct way to pass arguments

By value for simple types, by reference for modification

Apply modularity

Separate code into header and source files

Review Questions

1. What is the difference between a function declaration and definition?
2. When should parameters be passed by reference?
3. What advantages does modular code organization offer?

Ready for practical exercises? Let's create functions for working with arrays and mathematical calculations!