Functions

Functions allow you to group reusable blocks of code and break programs into logical parts.

1 What is a Function?

A function is a named block of code that performs a specific task. Functions help with code reuse, readability, and modularity.

2 Defining and Calling a Function

```cpp
void greet() {
    cout << "Hello!\n";
}

int main() {
    greet();  // function call
    return 0;
}
```

2.1 Functions with Parameters

Pass-by-Value: A copy of the variable is passed. Changes inside the function do **not** affect the original.

```cpp
void square(int x) {
    x = x * x;
    cout << "Inside function: " << x << endl;
}

int main() {
    int num = 5;
    square(num);
    cout << "After function: " << num << endl;  // still 5
}
```

Pass-by-Reference: The actual variable is passed using `&`. Changes inside the function **do** affect the original.

```cpp
void doubleValue(int &n) {
    n = n * 2;
    cout << "Inside function: " << n << endl;
}

int main() {
    int num = 5;
    doubleValue(num);
    cout << "After function: " << num << endl;  // now 10
}
```

Summary:

- Use **pass-by-value** when you don't want the function to change the original variable.
- Use **pass-by-reference** when the function should modify the original variable.

3 Function Prototypes

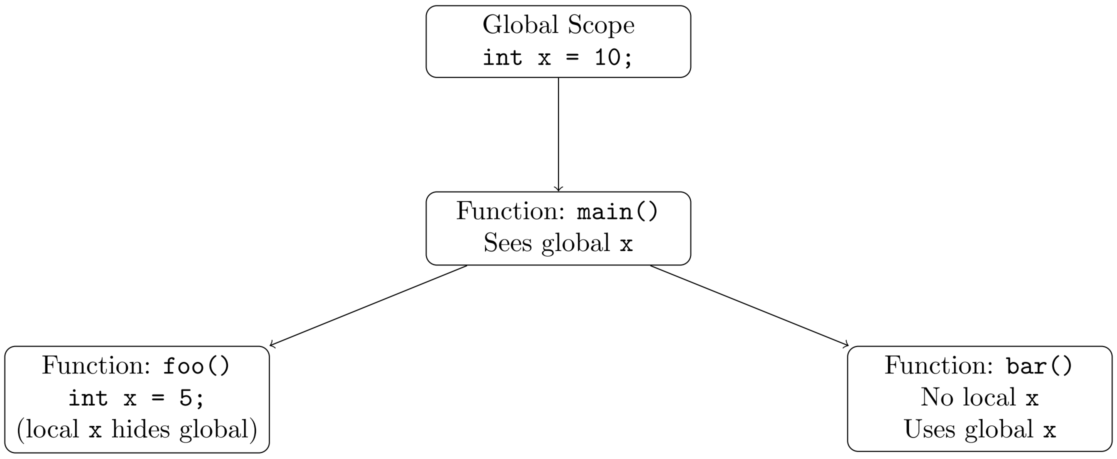
Prototypes tell the compiler about a function before its actual definition.

```
int add(int, int);  // prototype

int main() {
    cout << add(2, 3);
}

int add(int a, int b) {
    return a + b;
}
```

4 Scope of Variables

Variables can be declared globally (outside any function) or locally (inside a function). Local variables exist only within their function.

Scope Diagram:



5 Void vs Non-Void Functions

- `void` functions do not return a value. - Non-void functions specify a return type like `int` or `double`.

```
void sayHi() {
    cout << "Hi!\n";
}

int getFive() {
    return 5;
}
```