Functions and Member Functions

Defining reusable code: Global functions, Member functions, and the `inline` keyword.

Global Function Concepts

Function Structure and Execution

Function Definition

- **Function Declaration:** Tells the compiler about a function's name, return type, and parameters.
- **Function Definition:** Contains the actual block of code (the body) that executes the task.
- **Function Call:** Executes the code in the function's body.

```
int squareNum(int); // Declaration

int main() {
    int x = 5;
    int result = squareNum(x); // Call
    return 0;
}

int squareNum(int num) { // Definition
    return num * num;
}
```

Function Definitions in Detail

- **Definition Location:** Can be defined before or after the `main()` function.
- If defined **after** `main()`, a separate **declaration (prototype)** is mandatory before the function is called.
- The definition holds the compiler's main reference point for executing the program's logic.
- **Return Statement:** Terminates the function and returns control (and optionally a value) to the calling function.

```
int main() {
    int num = 10;
    cout << "10 squared is: " << squareNum(num) << endl;
    return 0;
}
// squareNum() must be defined somewhere else (e.g., a header or another file)
// or defined above main() if no prototype is used.
```

Member Functions

Defining Member Functions

- A member function is a function **declared inside a class** definition.
- It operates directly on the **data members** of the class object.
- It has access to **all** members (public, private, protected) of its own class.

Two ways to define them:

1. **Inside the Class** (Inline definition)
2. **Outside the Class** (Non-inline definition)

```
 class Rectangle {
private:
    double length;
    double width;
public:
    // Declaration inside the class (prototype)
    double calculateArea();

    // Inline Definition (Definition inside the class)
    void setData(double l, double w) {
        length = l;
        width = w;
    }
};
```

Definition **Outside** the Class

- Used for larger functions to keep the class definition clean and readable.
- Requires the **Scope Resolution Operator (`::`)** to link the function back to its class.
- The function must still be **declared** inside the class definition first.

Syntax:

```
Return_Type ClassName::FunctionName(parameters) { /* body */ }
```

```
// Continuation from previous slide's class declaration

double Rectangle::calculateArea() {
    return length * width;
}

int main() {
    Rectangle rect;
    rect.setData(10.0, 5.0);
    cout << "Area: " << rect.calculateArea();
    return 0;
}
```

The `inline` Keyword

`inline` Functions

- **Purpose:** Used as a request to the compiler to perform **inline expansion** instead of a regular function call.
- **Expansion:** The compiler replaces the function call with the actual function code at compile time.
- **Benefit:** Eliminates the overhead of a function call (stack setup, return address jump), resulting in faster execution.
- **Best Use:** **Small functions** (1-2 lines) where the overhead of the call is significant compared to the function's body.

```
 // Declaration/Definition must use the inline keyword
inline int Add(int a, int b) {
    return a + b;
}

int main() {
    int result = Add(2, 6);
    // Compiler replaces this line with:
    // int result = 2 + 6;
    return 0;
}
```

Rules for **`inline`** and Member Functions

Implicit Inline Rules

- ✓ **Any member function defined INSIDE** the class body is **implicitly treated** as an inline function by the compiler.
- ⊘ The `inline` keyword is **ignored** by the compiler if the function is too complex (e.g., loops, recursion, large size).

Inside vs. Outside Definition

Feature	Inside Class	Outside Class
Inline by Default	Yes	No
Resolution	Automatic	Requires `ClassName::`