COMSATS UNIVERSITY ISLAMABAD, ABBOTTABAD

Report on the Evolution Architecture

FAMIA MEHMOOD AND MALIAKA MEHMOOD

FA22-BSE-025

FA22-BSE-029

Report on the Evolution of the C Programming Language Architecture

1. Introduction

The C programming language, developed in the early 1970s by Dennis Ritchie at Bell Labs, has undergone significant evolution over the decades. It has become one of the most widely used programming languages, particularly in system programming, embedded systems, and application development. This report outlines the major releases of C, their features, architectural changes, and the evolution of its architecture from its inception to the latest standard, C18.

2EVOLUTION :

Key Architectural Models Applied Over Time

1. **Early C Architecture (1972)**
   - Initially, C was created as a systems programming language with **basic call and return** mechanisms (function calls).
   - Early C compilers had minimal abstraction, primarily working at a very low level.
2. **K&R C (1978)**
   - The architecture evolved with **function calls** and **stack-based memory management**.
   - Still no formal design patterns, but there was some separation of concerns (compiler vs. code).
3. **ANSI C (1983)**
   - **Modular Programming**: Header files became more prominent, establishing a basic separation of concerns.
   - Function calls and returns were better structured, with a focus on portability.
4. **C89 Standard (1989)**
   - **Type-based Architecture**: Stronger type checking and modularity.
   - Compilers started becoming more structured, and code was organized into clear **modules** and **libraries**.
5. **C99 (1999)**
   - **Pipe and Filter Architecture**: Functions started being treated more as filters (taking inputs, processing, and returning outputs). This allowed for cleaner, more modular code.
   - Use of **variable-length arrays** and **inline functions** made the code more flexible and reusable.
6. **C11 (2011)**
   - **Concurrency Architecture**: Support for **multi-threading** and atomic operations indicated the move toward more complex software architectures (e.g., client-server models, distributed systems).

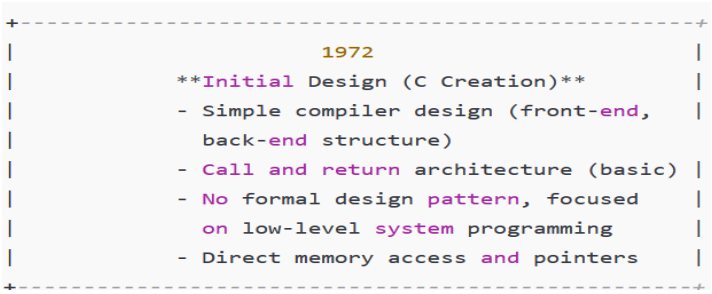
- Architectural models for **parallelism** emerged. C11 supported **shared-memory models**.

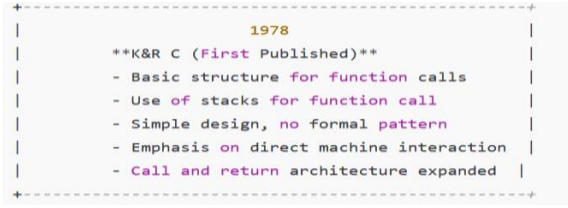
7. **C17 (2017)**
   - Minor refinements with focus on **bug fixes** and keeping the architecture backward-compatible while maintaining modern concurrency tools.
8. **C23 (2023)**
   - **Layered and Modular Architectures** continued to evolve.
   - Continued focus on **atomic operations** and **multi-threading**, with support for **designated initializers** and **enhanced memory management**.
   - More modern memory optimizations and optimizations for function calls.

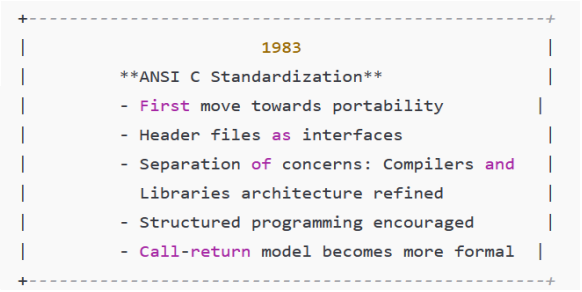
Key Architectural Patterns in C Language Evolution

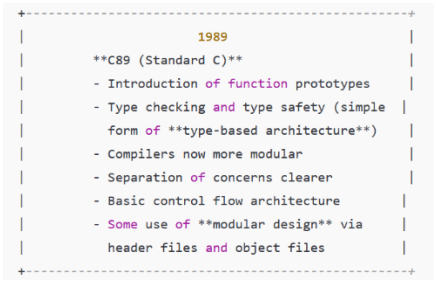
1. **Call and Return Architecture** (early stages):
   - Functions as the basic unit of abstraction, where function calls and returns managed program flow.
2. **Modular Architecture** (introduced in C89, refined in C99):
   - Emphasis on organizing code into distinct modules using **header files** and **source files**.
3. **Pipe and Filter Architecture** (C99 onward):
   - Functions treated as **filters** that accept input, process it, and return output, promoting reusable and modular code.
4. **Client-Server or Distributed System Architecture** (C11 onward):
   - With the introduction of **multi-threading**, C became more applicable for **distributed systems**, using separate threads and processes to handle different parts of a program concurrently.
5. **Layered Architecture** (C23 onward):
   - Focus on improving the **layered separation of concerns**, allowing code to be more modular, maintainable, and extensible.

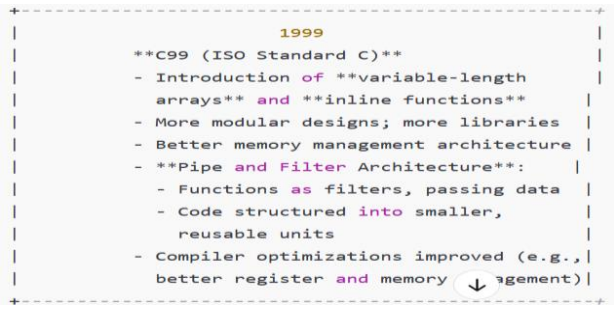
ARCHITECTURE DIAGRAM:

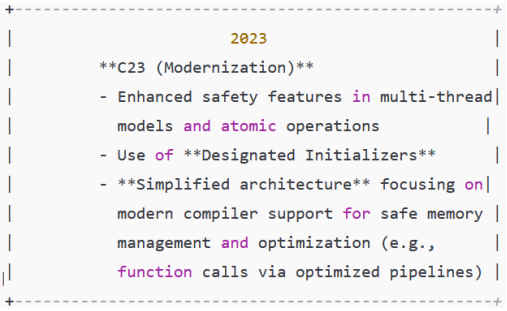












NOW IN 2024 C LANGUAGE IS FOLLOWING ARCHITECTURE :

+--+

| C Language in 2024 |

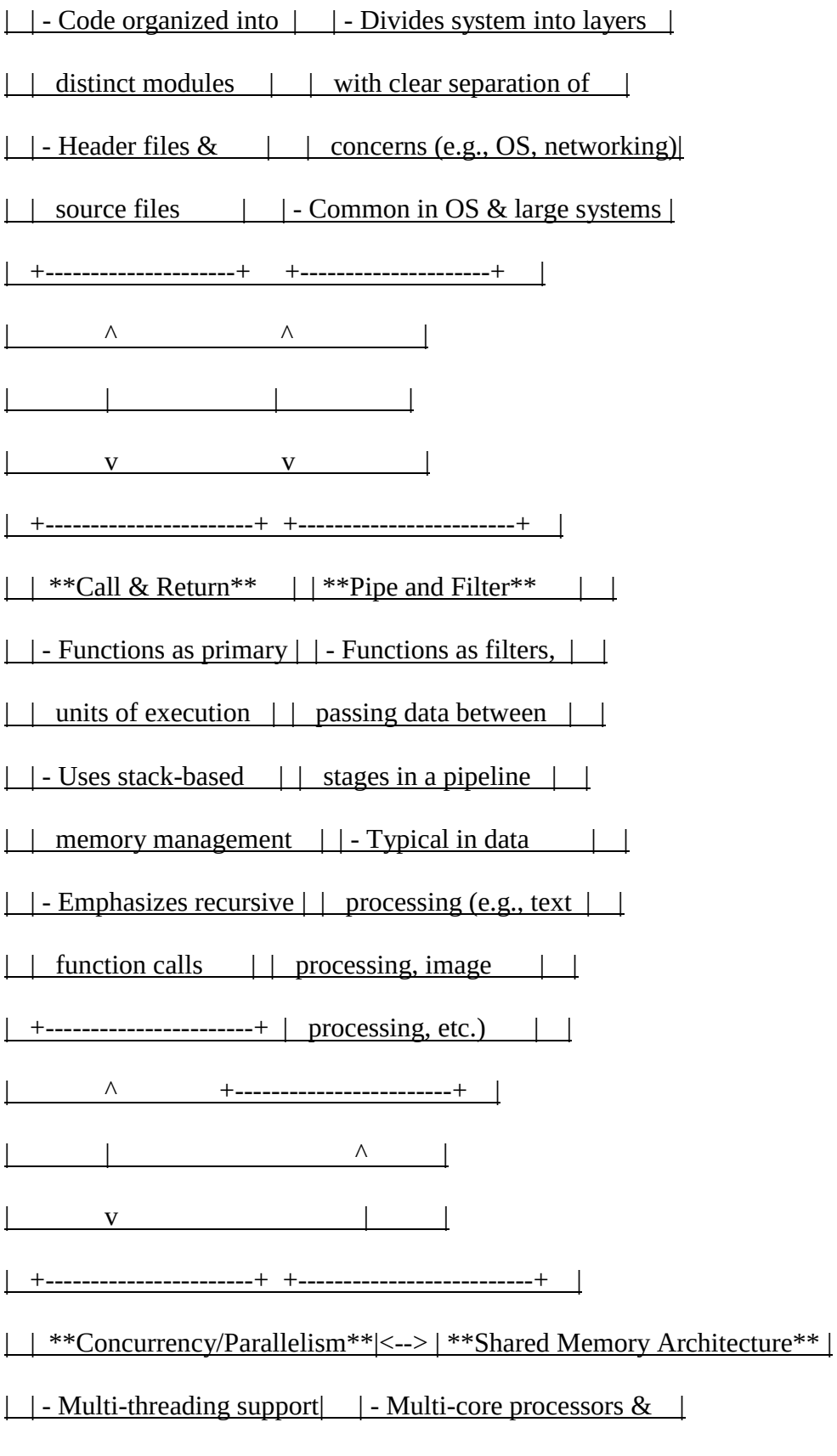
| (Modern Architectural Patterns) |

+--+

|_____|

| +--------------------+ +--------------------+ |

| | **Modular Architecture**|<--> | **Layered Architecture** |


- Code organized into distinct modules
- Header files & source files
- Common in OS & large systems
- Divides system into layers with clear separation of concerns (e.g., OS, networking)
Call & Return
- Functions as primary units of execution
- Uses stack-based memory management
- Emphasizes recursive function calls
Pipe and Filter
- Functions as filters, passing data between stages in a pipeline
- Typical in data processing (e.g., text processing, image processing, etc.)
Concurrency/Parallelism
- Multi-threading support
Shared Memory Architecture
- Multi-core processors &

| | - Atomic operations for | | parallel execution |

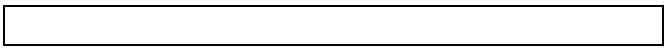
| | memory safety | | - Thread synchronization & |

| | - Modern OS/Server apps | | atomic operations |

| | using parallel tasks | | (e.g., multi-threaded systems)|

| +----------+ +----------+ |

|_____|



Visual representation:

[MODULAR ARCHITECTURE] <--> [LAYERED ARCHITECTURE]

^ ^

| |

v **v**

[CALL & RETURN] <--> [PIPE AND FILTER]

^ ^

| |

v **v**

[CONCURRENCY/PARALLELISM] <--> [SHARED MEMORY ARCHITECTURE]

Key Notes:

- **Modular and Layered Architectures** are the primary frameworks for organizing large systems, especially operating systems, device drivers, and network stacks.
- **Concurrency/Parallelism** and **Shared Memory** are essential for performance in modern multi-core systems and multi-threaded applications.

- **Call and Return** (function-based execution) remains the backbone of C programming, ensuring clarity, modularity, and control flow.
- **Pipe and Filter** is heavily used in data processing and system tools, such as **Unix utilities**.

Conclusion:

In **2024**, C language follows these key architectures, allowing it to stay relevant for a wide range of applications from **embedded systems** to **operating systems** and **high-performance computing**. By incorporating modern concurrency tools, memory safety, and modular programming practices, C remains a powerful language for low-level and high-performance tasks.

OVERALL Conclusion:

The **C language architecture** evolved from a simple call-return based low-level system programming model to more modern architectural paradigms like **modular design**, **pipe and filter**, and **concurrent programming**. Each version of C incorporated features that aligned with growing software engineering trends and architectural needs, such as **modularity** in C89, **pipe-and-filter** patterns in C99, and **multi-threading** and **atomic operations** in C11 and beyond.

THE END