**Chapter 2**

**Week 1.2**

**7-11.The Beginning of Fortran**  
The Fortran (FORmula TRANslation) design project was initiated in **1954** by IBM under the leadership of John Backus. At the time, the primary goal was to create a high-level programming language that could simplify the development of scientific and mathematical programs. Fortran aimed to offer an alternative to the labor-intensive process of writing assembly code while achieving similar levels of efficiency.

**Primary Application Area of Computers During Fortran’s Design**  
When Fortran was designed, the primary application area for computers was **scientific and engineering computations**. These domains demanded significant mathematical and numerical calculations, such as missile trajectory analysis, atomic energy research, and solving complex differential equations. Fortran was specifically tailored to meet these needs by providing robust support for floating-point arithmetic, arrays, and matrices.

Control Flow Statements in Fortran I  
The control flow statements in Fortran I were directly inspired by assembly language constructs, as the early computing era heavily relied on assembly for programming. Basic constructs like the IF statement for conditional branching and the DO loop for iteration were modeled after the low-level operations available on hardware, providing a familiar structure for programmers transitioning from assembly to high-level programming.

The Transition to Fortran II  
The most significant advancement in Fortran II was the introduction of subprograms, which allowed developers to define reusable code blocks (functions or procedures). This feature enhanced the language's modularity, making it possible to break down complex programs into smaller, manageable components. Subprograms revolutionized programming by promoting code reuse and simplifying the debugging process, thereby broadening Fortran's applicability.

Fortran IV to Fortran 77: New Control Flow Statements  
The transition from Fortran IV to Fortran 77 marked a significant leap in programming language design with the addition of several new control flow statements. These included:

IF-ELSE-ENDIF: This structured decision-making construct improved program readability and maintainability by clearly separating conditions and their corresponding actions.

DO WHILE: This looping construct allowed conditional iteration, enabling programs to execute loops while specific conditions were met.

BLOCK IF: This provided enhanced control over nested conditionals, offering a more organized approach to handling multiple levels of decisions within programs. These features reflected the influence of structured programming principles, which emphasized readability, modularity, and reduced complexity in program design.

23-25. **The Beginning of COBOL**  
The design process for COBOL (Common Business-Oriented Language) began in **1959**. This effort was initiated under the guidance of the Short-Range Committee, a group assembled by the U.S. Department of Defense (DoD). The primary goal was to create a universal business language that could process large amounts of data and be easily understood by individuals without extensive programming expertise. The collaborative effort involved representatives from government, academia, and private industries, ensuring a wide range of perspectives in the language's design.

**Plankalkül's Influence on COBOL**  
One significant data structure introduced in COBOL that originated with Konrad Zuse's **Plankalkül** was the concept of **hierarchical data structures**, or **records**. This innovation allowed COBOL to represent complex data relationships effectively, organizing data into fields and records. For example, a payroll system could group employee data such as name, ID, and salary into a single, structured entity. This feature became foundational for COBOL’s ability to handle business data efficiently and set a standard for future programming languages.

**The Role of the U.S. Department of Defense**  
The U.S. Department of Defense (DoD) played a pivotal role in COBOL's early success by mandating its use in government systems. This requirement ensured that COBOL gained widespread adoption across various industries, as companies providing services to the government also needed to adopt the language. The DoD's influence was instrumental in establishing COBOL as the primary language for business applications during its early years. Its readability, support for decimal arithmetic, and ability to process structured data made COBOL particularly well-suited for financial systems, payroll processing, and other enterprise-level applications.

54. Python uses **lists** in place of traditional arrays found in languages like C or Java. Unlike arrays, Python lists are dynamic, meaning they can grow or shrink in size as needed, and they can hold elements of mixed data types. This flexibility makes lists a versatile and powerful tool for developers. Lists in Python support a wide range of operations, such as slicing, concatenation, and built-in methods like append, remove, and sort.

57. The **switch statement** in C# addresses several deficiencies of the traditional switch statement in C, enhancing both functionality and readability. In C, the switch statement is limited to primitive data types, requires explicit break statements to prevent fall-through, and cannot return values directly.