- Reading Comprehension
  - Compilers and Programming Environments for C
  - Programming Environments for C:**
  - The C Development Cycle
  - Steps to Build a C Program:**
  - Difference between Assembler and Compiler
  - Difference between Compiler and Interpreter
  - Difference between Syntax and Semantics
  - Flowchart, Algorithms, and Pseudocode

Reading Comprehension

How to Get Started with C

To get started with C programming, follow these steps:

1. **Learn the Basics**:
2. **Set Up Your Environment**:
3. **Write Your First Program**:
4. **Practice Regularly**:
5. **Utilize Learning Resources**:

Compilers and Programming Environments for C

As a beginner learning about C compilers, what you should know:

▶ 🗒️ Answer

1. **Understanding the Concept of a compiler**: A compiler is a software tool that translates human-readable source code written in C into machine-readable binary code that computers can execute.

2. **Choosing a Compiler**: GCC (GNU Compiler Collection) is a popular and widely-used compiler for C programming. It's freely available and supports multiple platforms, making it ideal for beginners.

3. **Setting Up the Environment**: Install GCC on your computer. On Linux systems, it often comes pre-installed. For Windows, you can use MinGW (Minimalist GNU for Windows) or Cygwin. Follow installation instructions provided by the compiler's documentation.

What are the components of a C compiler?:

▶ 🗒️ Answer

The components of a C compiler typically include:

1. **Preprocessor**: This component processes directives like `#include` and `#define` before the actual compilation begins. It prepares the source code for compilation by removing comments, handling macros, and including header files.

2. **Compiler**: The compiler translates the preprocessed source code into assembly code or intermediate representation. It checks syntax and semantics, generates intermediate code, and performs optimizations.

3. **Assembler**: This component translates the intermediate representation or assembly code into machine code specific to the target architecture. It converts human-readable assembly instructions into binary machine language instructions.

4. **Linker**: The linker combines object files generated by the compiler along with any necessary libraries to create an executable file. It resolves external references between different modules and produces a complete program ready for execution.

> These components work together in a sequence to translate C source code into machine-executable instructions.

Discuss popular C compilers and their features:

▶ 🗒️ Answer

Here are some notable compilers and their key features:

1. **GCC (GNU Compiler Collection)**:
    - Widely used for its robustness and portability.
    - Supports various programming languages, including C, C++, and Fortran.
    - Provides extensive optimization options for improving code performance.
    - Offers support for multiple platforms, including Linux, Windows, and macOS.

2. **Clang**:

    - Known for its fast compilation speed and diagnostics.
    - Compatible with LLVM (Low-Level Virtual Machine) infrastructure, enabling advanced optimizations.
    - Supports modern C and C++ language features.
    - Integrated with tools like Clang Static Analyzer for identifying bugs and security vulnerabilities.

3. **Microsoft Visual C++ Compiler**:

    - Specifically designed for Windows development.
    - Provides integration with Visual Studio IDE, offering a comprehensive development environment.
    - Supports Microsoft-specific features like Windows Runtime and .NET Framework.
    - Offers optimizations for Windows platforms and compatibility with Microsoft's development tools.

4. **Intel C Compiler**:

    - Known for its performance optimizations on Intel processors.
    - Utilizes advanced vectorization and parallelization techniques for enhanced performance.
    - Supports various optimization levels tailored for different target architectures.
    - Integrated with Intel Parallel Studio for parallel programming and performance analysis.

5. **Tiny C Compiler (TCC)**:

    - Focused on producing small and fast executables.
    - Suitable for embedded systems and resource-constrained environments.
    - Offers quick compilation times and minimal dependencies.
    - Supports most of the C99 standard and some C11 features.

> These compilers cater to diverse requirements, ranging from general-purpose development to platform-specific optimizations and specialized use cases.

How do integrated development environments (IDEs) aid in C programming?:

▶ 📄 Answer

Integrated Development Environments (IDEs) offer several benefits for C programming:

1. **Code Editing**: IDEs provide advanced code editors with features like syntax highlighting, code completion, and automatic indentation, making it easier to write and maintain C code.

2. **Debugging Tools**: IDEs offer built-in debuggers that allow developers to set breakpoints, inspect variables, and step through code execution, helping to identify and fix errors more efficiently.

3. **Project Management**: IDEs help organize C projects by providing tools for creating, managing, and navigating project files and directories, which is particularly useful for larger codebases.

4. **Version Control Integration**: Many IDEs integrate with version control systems like Git, allowing developers to commit, pull, push, and merge changes directly from within the IDE, facilitating collaboration and code management [tech/devops/articles].

5. **Compiler Integration**: IDEs often come with integrated compilers or support for external compilers, enabling developers to compile and build C programs within the same environment, with options for customizing build configurations.

6. **Code Refactoring**: IDEs offer tools for refactoring C code, such as renaming variables, extracting methods, and finding and fixing code smells, helping to improve code quality and maintainability.

> Overall, IDEs streamline the C programming workflow by providing a centralized environment with essential tools and features, enhancing productivity and code quality.

Programming Environments for C:**

How does the choice of programming environment impact C development?:

▶ 🗒️ Answer

The choice of programming environment significantly impacts C development in several ways:

1. **Productivity**: A well-designed environment enhances productivity by providing features like code completion, syntax highlighting, and project management tools, allowing developers to write code more efficiently.
2. **Debugging**: Integrated debugging tools streamline the process of identifying and fixing bugs, reducing development time and improving software quality.
3. **Collaboration**: Environments with version control integration facilitate collaboration among team members, enabling efficient code sharing, reviewing, and merging.
4. **Compiler Integration**: Seamless integration with compilers simplifies the build process, allowing developers to compile and execute their code within the same environment.
5. **Code Refactoring**: Environments offering code refactoring tools help improve code maintainability by assisting in restructuring and optimizing existing codebases.
6. **Learning Curve**: The complexity of the environment may affect developers' learning curves, with more feature-rich environments potentially requiring more time to master.
7. **Community Support**: Environments with active communities often provide extensive documentation, tutorials, and plugins, enhancing the overall development experience.
8. **Platform Compatibility**: Some environments may be platform-specific, limiting the portability of developed code across different operating systems.

> Choosing the right programming environment tailored to the project's requirements and developers' preferences can significantly impact the efficiency, quality, and success of C development projects.

What features should be considered when selecting a programming environment for C?:

▶ 🗒️ Answer

When selecting a programming environment for C development, several key features should be considered:

1. **Code Editing**: Look for an environment with robust code editing capabilities, including syntax highlighting, code completion, and intelligent code navigation.
2. **Debugging Tools**: Choose an environment that offers advanced debugging tools, such as breakpoints, watchlists, and real-time variable inspection, to facilitate efficient debugging processes.
3. **Project Management**: Opt for an environment with project management features, like integrated task trackers, project templates, and build automation tools, to streamline project organization and collaboration.
4. **Version Control Integration**: Ensure the environment supports seamless integration with version control systems like Git, enabling efficient collaboration, code versioning, and change tracking.
5. **Compiler Integration**: Look for environments that integrate well with C compilers, providing features like automatic code compilation, error checking, and optimization options.
6. **Code Refactoring**: Consider environments that offer code refactoring tools, such as automated code restructuring and renaming, to improve code readability, maintainability, and efficiency.

🌐 Sources

1. Aalpha.net - Factors to Consider When Choosing a Programming Language
2. LinkedIn - How to Choose a Programming Language for Computer
3. GeeksforGeeks - How to Choose a Programming Language For a Project?
4. Homework.Study.com - Describe four different considerations when choosing a
5. Quora - What are things to consider when you select a programming language
6. Divisionartistcom - Factors influencing the choice of programming languages

Discuss the advantages and disadvantages of different programming environments for C:

▶ 📄 Answer

When considering different programming environments for C, it's essential to weigh their advantages and disadvantages:

1. **Text Editors**:

- *Advantages*: Lightweight, versatile, and customizable. Ideal for quick edits and small projects.
- *Disadvantages*: Lacks advanced features like debugging tools and project management capabilities.

2. **Integrated Development Environments (IDEs)**:

   - *Advantages*: Offers comprehensive features like code editing, debugging tools, project management, and version control integration. Enhances productivity and facilitates complex project development.
   - *Disadvantages*: Can be resource-intensive and overwhelming for beginners. Some IDEs may have a steep learning curve.

3. **Command-Line Interfaces (CLIs)**:

   - *Advantages*: Lightweight, efficient, and suitable for experienced developers who prefer minimalistic setups. Provides direct control over compilation and execution processes.
   - *Disadvantages*: Lacks graphical interface and may require manual configuration. Less beginner-friendly compared to IDEs.

4. **Online Compilers**:

   - *Advantages*: Convenient for quick prototyping and sharing code. No installation required, accessible from any device with internet access.
   - *Disadvantages*: Limited functionality compared to desktop IDEs. Security concerns regarding data privacy and reliability of online services.

> Consider these factors to choose the most suitable programming environment based on your project requirements, skill level, and personal preferences.

The C Development Cycle

What are the stages involved in the C development cycle?

▶ 🗒️ Answer

The stages involved in the C development cycle typically include:

1. **Problem Definition**: Identifying and understanding the problem that the program is intended to solve.
2. **Problem Analysis**: Analyzing the problem to determine its requirements, constraints, and potential solutions
3. **Algorithm Development**: Creating an algorithm or step-by-step procedure to solve the problem, often using flowcharts, pseudocode, or other design tools.
4. **Coding & Documentation**: Writing the actual C code to implement the algorithm, along with documenting the code to make it understandable and maintainable.
5. **Testing & Debugging**: Testing the program to ensure that it behaves as expected, identifying and fixing any errors (bugs) encountered during testing.

> These stages are iterative and may require revisiting previous stages as the development process progresses. Additionally, tasks such as optimization, deployment, and maintenance may be part of the overall development cycle depending on the project's requirements and complexity.

🌐 Program Development Life Cycle

1. C - program development cycle
2. What is program development cycle in C language?
3. C Tutorials - Program Development Life Cycle

How does the development cycle differ between small and large-scale C projects?

▶ 🗒️ Answer

The development cycle for small and large-scale C projects differs primarily in terms of complexity, duration, and team size:

1. **Scope and Complexity**:
    - Small-scale projects typically have simpler requirements and can be completed relatively quickly, often within weeks to months.
    - Large-scale projects involve more complex requirements, extensive planning, and longer development times, often spanning months to years.
2. **Team Size**:
    - Small-scale projects may involve a small team or even a single developer, leading to more direct communication and collaboration.

- Large-scale projects typically require a larger development team, with various roles such as architects, developers, testers, and project managers. This necessitates more structured communication and coordination among team members.

3. **Development Process**:
   - Small-scale projects may follow a more straightforward development process with fewer formalities and documentation requirements.
   - Large-scale projects often adhere to more rigorous methodologies such as Agile or Waterfall, involving comprehensive planning, design, implementation, testing, and deployment phases.

4. **Resource Allocation**:
   - Small-scale projects may have limited resources allocated for development, leading to a focus on essential features and efficiency.
   - Large-scale projects often require more significant investments in resources, including personnel, time, and budget, to address the complexity and scale of the project.

In summary, while both small and large-scale C projects follow similar development principles, the scale and complexity of the project significantly influence the development cycle's duration, team dynamics, and resource allocation.

🌐 Sources

1. Comparison between Small and Large Projects
2. (PDF) Comparing Methods for Large-Scale Agile Software Development
3. Categories of Project

What tools and techniques are commonly used to streamline the C development process?

▶ 🗒️ Answer

To streamline the C development process, various tools and techniques are commonly employed:

1. **Integrated Development Environments (IDEs)**: IDEs like Visual Studio, Eclipse, and JetBrains CLion offer features such as code editing, debugging, and version control integration, enhancing productivity.

2. **Version Control Systems (VCS)**: Tools like Git, SVN, and Mercurial enable team collaboration, code tracking, and managing project versions, ensuring code integrity and facilitating collaboration.

3. **Build Automation Tools**: Tools such as Make, CMake, and GNU Autotools automate the compilation, testing, and deployment processes, reducing manual effort and minimizing errors.

4. **Code Linters and Static Analyzers**: Tools like Clang-Tidy and Coverity perform static code analysis to identify potential bugs, code quality issues, and adherence to coding standards, improving code reliability and maintainability.

5. **Continuous Integration/Continuous Deployment (CI/CD)**: CI/CD pipelines, implemented using tools like Jenkins, Travis CI, and GitLab CI/CD, automate code integration, testing, and deployment, ensuring rapid feedback and consistent delivery.

6. **Documentation Tools**: Documentation tools like Doxygen help in generating project documentation from source code comments, ensuring comprehensive and up-to-date project documentation.

7. **Code Review Tools**: Platforms like GitHub, GitLab, and Bitbucket provide built-in code review features, facilitating peer code reviews, feedback, and collaboration.

> By leveraging these tools and techniques, C developers can streamline their development process, improve code quality, and accelerate project delivery.

🌐 Sources

1. How to Streamline Your Software Development Process
2. DevOps Streamlining Software Development Processes
3. Understanding Software Development: Process, Tools & Practice
4. 6 Key Strategies to Streamline Software
5. The 7 Best AI Tools for Programmers to Streamline Development in 2024

Steps to Build a C Program:**

Can you outline the process of writing, compiling, and executing a simple C program?

▶ 🗒️ Answer

Here's the process of writing, compiling, and executing a simple C program:

1. **Writing the Program**: Use a text editor to write your C program code. Save the file with a `.c` extension, such as `example.c`.

2. **Compiling the Program**: Open a terminal or command prompt and navigate to the directory where your C program file is saved. Use a C compiler like GCC to compile the source code into an executable file. For example, you can use the command `gcc example.c -o example` to compile `example.c` into an executable named `example`.

3. **Executing the Program**: After successful compilation, you can run the executable file. In the terminal or command prompt, type the name of the executable file (without the extension) and press Enter. For example, if your executable is named `example`, type `./example` and press Enter. This will execute your C program.

🌐 Sources

1. Compilation In C | Complete Explanation (+Examples)
2. Compiling a C Program: Behind the Scenes
3. Compilation Process in C - Scaler Topics

Are there any best practices or conventions to follow when structuring a C program?

▶ 🗒️ Answer

Here are some best practices and conventions for structuring a C program:

1. **Modularization**: Break your program into smaller, manageable modules or functions. This promotes code reuse and makes your code easier to understand and maintain.
2. **Naming Conventions**: Follow consistent and descriptive naming conventions for variables, functions, and other identifiers. Use meaningful names that reflect the purpose and functionality of the elements.
3. **Header Files**: Use header files to declare function prototypes, macros, and type definitions. Separate interface declarations from implementation details to improve code readability and maintainability.

4. **Directory Structure**: Organize your source files and headers into logical directories based on functionality or module. This helps in managing larger projects and makes it easier to navigate through the codebase.
5. **Comments and Documentation**: Include clear and concise comments to explain complex algorithms, data structures, and non-obvious code segments. Documenting your code enhances its understandability and aids future maintenance.
6. **Consistent Formatting**: Adopt a consistent coding style and formatting throughout your codebase. Consistency improves readability and reduces the likelihood of errors.

🌐 Sources

1. C Style Guide
2. Recommended C Style and Coding Standards
3. How to structure a C program effectively
4. C Coding Standard
5. Best practices for folder structures of C projects containing hundreds of source code files?
6. What are some C programming best practices?

Difference between Assembler and Compiler

Explain the difference between an assembler and a compiler:

▶ 🗒️ Answer

An assembler and a compiler are both language processors, but they serve different purposes and operate at different levels of abstraction:

1. **Assembler**:
    - An assembler is a program that translates assembly language code into machine code.
    - It converts mnemonic instructions and symbolic names into binary machine language instructions.

- Assemblers are specific to the architecture of the target machine and directly produce machine code.
- They work at a low level, dealing with individual instructions and memory addresses.

2. **Compiler**:

   - A compiler is a more sophisticated program that translates high-level programming language code into machine code or intermediate code.
   - It processes entire source code files and produces executable programs or intermediate representation.
   - Compilers are platform-independent and can generate code for multiple target platforms from the same source code.
   - They work at a higher level of abstraction, translating complex language constructs into equivalent machine code instructions.

> In summary, the main difference lies in their input and output: assemblers take assembly language as input and produce machine code, while compilers take high-level programming language code as input and produce machine code or intermediate code.

🌐 Sources

1. Difference Between Compiler and Assembler - BYJU'S
2. Difference between Compiler and Assembler - GeeksforGeeks

How do they translate source code into machine code?:

▶ 🗒️ Answer

Translating source code into machine code involves several steps performed by a compiler:

1. **Lexical Analysis**:

   - The compiler breaks down the source code into tokens such as keywords, identifiers, literals, and operators.

2. **Syntax Analysis (Parsing)**:

   - The compiler checks the arrangement of tokens to ensure they conform to the syntax rules of the programming language.

3. **Semantic Analysis**:
    - The compiler verifies the semantics of the code, ensuring it makes logical sense according to the language rules.
4. **Intermediate Code Generation**:
    - Some compilers generate an intermediate representation of the source code for optimization or portability purposes.
5. **Optimization**:
    - The compiler applies various optimization techniques to improve the efficiency and performance of the generated code.
6. **Code Generation**:
    - Finally, the compiler translates the optimized intermediate code or the source code directly into machine code instructions specific to the target architecture.
7. **Linking (optional)**:
    - In some cases, the compiler may also perform linking, which involves combining multiple object files and libraries into a single executable file.

This process ensures that the source code is transformed into machine-readable instructions that can be executed by the computer's CPU.

🌐 Sources

1. What is a compiler? How source code becomes machine ... - InfoWorld
2. How does a compiler generate machine code? - LinkedIn

Difference between Compiler and Interpreter

Compare and contrast compilers and interpreters:

▶ 🗒️ Answer

Here's a comparison between compilers and interpreters:

1. **Compilation Process**:
    - Compiler: Translates the entire source code into machine code before execution.
    - Interpreter: Executes code line by line, translating and executing each statement as it encounters them.
2. **Performance**:
    - Compiler: Generally produces faster-running code as it optimizes the entire program before execution.
    - Interpreter: May have slower performance as it translates and executes code in real-time.
3. **Error Detection**:
    - Compiler: Identifies errors in the entire program at once, providing comprehensive feedback.
    - Interpreter: Detects errors as it executes the code, stopping at the first encountered error.
4. **Memory Usage**:
    - Compiler: Usually generates standalone executable files, requiring less memory during execution.
    - Interpreter: Requires additional memory as it must remain active during code execution.
5. **Ease of Implementation**:
    - Compiler: More complex to implement due to the need to optimize the entire codebase.
    - Interpreter: Relatively simpler to implement, as it executes code line by line.

> In summary, compilers translate the entire source code into machine code before execution, resulting in potentially faster-running code but requiring more memory. On the other hand, interpreters execute code line by line, making error detection easier but potentially leading to slower performance. The choice between them depends on factors like performance requirements, ease of implementation, and debugging needs.

What are the implications for program execution and performance?

▶ 🗒️ Answer

The implications for program execution and performance are significant and can impact various aspects of software development and system operation:

1. **Speed and Efficiency**: The performance of a program determines how quickly and efficiently it can execute tasks and process data.
2. **Resource Utilization**: Efficient program execution optimizes resource usage, including CPU, memory, and disk space, leading to better overall system performance.
3. **User Experience**: Faster program execution enhances user satisfaction by reducing waiting times and improving responsiveness.
4. **Scalability**: Programs with good performance can handle increasing workloads and scale effectively without degradation in execution speed.
5. **Reliability**: Well-performing programs are less prone to errors and crashes, contributing to system stability and reliability.
6. **Energy Consumption**: Efficient program execution minimizes energy consumption, which is crucial for battery-powered devices and environmentally sustainable computing.
7. **Competitive Advantage**: High-performance software can provide a competitive edge in various industries by enabling faster innovation, better customer service, and cost savings.
8. **Strategic Execution**: Program management facilitates strategic execution by aligning program goals with organizational objectives and ensuring performance metrics are met. Understanding and optimizing program execution and performance are essential for achieving desired outcomes in software development and system operation.

🌐 Sources

1. How to explain to someone that the execution performance in programming isn't everything - Quora
2. Do Operating Systems slow down program execution? - Stack Overflow
3. Performance of Computer in Computer Organization - GeeksforGeeks
4. Program Management: The Key to Strategic Execution - Planview
5. Program Execution and Project Management - Seam Group
6. Performance Implication - ScienceDirect

Difference between Syntax and Semantics

Define syntax and semantics in the context of programming languages:

▶ 🗒️ Answer

1. **Syntax**:
    - Syntax refers to the set of rules that define the structure and composition of valid statements or expressions in a programming language.
    - It governs how symbols, keywords, and punctuation are combined to form correct code constructs.
    - Syntax errors occur when code violates these rules, such as missing semicolons, mismatched parentheses, or incorrect keyword usage.
2. **Semantics**:
    - Semantics pertains to the meaning or interpretation of code constructs in a programming language.
    - It defines the behavior and functionality of the code, including its intended actions and outcomes when executed.
    - Semantics errors occur when code produces unintended results due to incorrect logic or misunderstanding of language constructs, even if the syntax is correct.
3. **Relation to Program Correctness**:
    - Syntax and semantics are both essential for ensuring program correctness.
    - Correct syntax allows the program to be compiled or interpreted without errors, but it does not guarantee logical correctness.
    - Semantics ensures that the program performs the intended operations and produces the desired results.
    - Together, syntax and semantics contribute to program correctness by ensuring that code is both syntactically valid and logically sound.

> Syntax and semantics work together to ensure that programming languages are both grammatically correct and logically meaningful, contributing to the accuracy and effectiveness of software development.

🌐 Sources

1. Difference Between Syntax and Semantics - GeeksforGeeks
2. Programming Languages: Syntax, Semantics, Types - StudySmarter
3. Difference Between Syntax and Semantics - Tutorialspoint

Provide examples illustrating the difference between syntax and semantics errors:

▶ 🗒️ Answer

1. **Syntax Error**:
    - Syntax errors occur when code violates the grammatical rules of the programming language.
    - They prevent the program from compiling due to incorrect syntax.
    - Example:

```
// Syntax Error: Missing semicolon at the end of the statement
int main() {
    printf("Hello, world!")
    return 0;
}
```

    This code will result in a syntax error because the printf statement lacks a semicolon at the end.

2. **Semantic Error**:
    - Semantic errors occur when code is syntactically correct but produces unintended results due to logical mistakes.
    - These errors may lead to runtime issues or incorrect program behavior.
    - Example:

```
// Semantic Error: Incorrect mathematical operation
int main() {
    int a = 10;
    int b = 0;
    int result = a / b;
    printf("Result: %d\n", result);
```

```
    return 0;
}
```

This code compiles without errors, but it will cause a runtime error (division by zero) because it attempts to divide by zero, which is a semantic error.

3. **Corrected Code**:
    - Here's the corrected version of the code:

```
// Corrected Code
int main() {
    printf("Hello, world!");
    return 0;
}
```

This code fixes the syntax error by adding a semicolon at the end of the printf statement.

🌐 Sources

1. Understanding Syntax and Semantic Errors in C Programming - GeeksforGeeks

Flowchart, Algorithms, and Pseudocode

What are flowcharts, algorithms, and pseudocode?:

▶ 📄 Answer

1. **Flowcharts**:
    - Flowcharts are graphical representations of a process or algorithm, using various symbols to illustrate the steps involved.
    - They visually depict the flow of control or data through a system.
    - Flowcharts help in understanding, analyzing, and communicating complex procedures or algorithms.
    - Example: Flowcharts vs Pseudocode vs Code for Algorithms - LinkedIn.
2. **Algorithms**:

- Algorithms are step-by-step procedures or instructions for solving a problem or accomplishing a task.
- They provide a systematic approach to problem-solving and are often expressed in various forms, including pseudocode and flowcharts.
- Algorithms can be implemented in programming languages to automate tasks.
- Example: Algorithms - Edexcel - GCSE Computer Science Revision

3. **Pseudocode**:

- Pseudocode is a high-level description of a computer algorithm using natural language mixed with some programming language constructs.
- It provides a detailed yet understandable outline of the logic behind an algorithm without being tied to any specific programming language syntax.
- Pseudocode helps in planning and designing algorithms before actual coding.
- Example: Pseudocode and Flowcharts - Codecademy

Example Pseudocode:

```c
#include <stdio.h>

// Pseudocode:
// 1. Start
// 2. Initialize variables n and factorial
// 3. Set n to 5
// 4. Set factorial to 1
// 5. Start loop from i=1 to n
//    a. Update factorial by multiplying it with i
// 6. End loop
// 7. Print factorial
// 8. End

int main() {
    // Algorithm:
    // 1. Initialize variables n and factorial to 5 and 1, respectively
    // 2. Start loop from i=1 to n
    //    a. Update factorial by multiplying it with i
    // 3. End loop
    // 4. Print factorial
    // 5. End

    int n = 5;
    int factorial = 1;

    // Flowchart:
    // [START] --> [Initialize variables n and factorial]
    //           --> [Start loop]
```

```
    //               --> [Update factorial]
    //           --> [End loop]
    //           --> [Print factorial]
    // [END]

    for (int i = 1; i <= n; ++i) {
        factorial *= i;
    }

    printf("Factorial of %d = %d\n", n, factorial);

    return 0;
}
```