02_01_Why C?

Why *C* ?

- Many situations where it is *only* language or system available
  - » Small, embedded systems, instrumentation, etc.
- Many “low-level” situations that don’t have support for “high-level” languages
  - » Operating systems, real-time systems, drivers

A Short History

- In the beginning …
  - » Machine language
  - » Assembly language

Really primitive!
Too difficult for big projects

- So “high level” languages were invented
  - » Non-recursive:– Fortran, Cobol
  - » Recursive:– Algol, Lisp, Snobol, PL/1, etc.

Too advanced!
Too *much* infrastructure for operating systems, control systems, many kinds of projects

A Short History (continued)

- 1960s:– “system programming” languages invented
  - » More direct control over hardware
  - » Not so primitive as Assembly Language

- E.g., a veritable alphabet soup
  - » BLISS, BCPL, PL/360, …

A Short History (continued)

- Late 1960s:– MIT's *MULTICS* project doesn't live up to expectation
  - » Bell Labs wants to pull out
  - » Ken Thompson starts on *Unix* as alternative
- Thompson creates own system language
  - » Based on BCPL, called *B*
  - » First version of Unix written in *B*
- Unix takes hold inside Bell Labs
  - » Needs to be re-written
  - » Dennis Ritchie *et al* create *C* as typed successor to *B*

A Short History (continued)

- ***C*** wins the race of "system programming" languages
  - » Language of choice for many operating systems
  - » Many applications

- Object-oriented programming begins to emerge
  - » Stroustrup starts on *C with Classes in 1979*
  - » Backward compatible with *C*
  - » Later renamed C++

A Short History (continued)

- By 1990, *C++* is language of choice for many kinds of applications
  - » *Good:–* object-oriented, classes, inheritance, etc.
  - » *Bad:–* all of the ugly characteristics of *C* still persist
  - » Complex object model, memory mgt, etc.
  - » officially standardized in 1998

- Jim Gosling at Sun Microsystems starts a complete rewrite of *C++*
  - » Simpler object model
  - » Cleaner
  - » Portable, can be embedded in web pages, etc.
  - » Eventually called *Java*

Why it is called C?

- It was named "C" because its features were derived from an earlier language called "B“ ☺
- According to Ken Thompson was a stripped-down version of the BCPL programming language'.

Built for Performance

- C is widely used to develop systems that demand performance, such as operating systems, embedded systems, real-time systems and communications systems.

Application	Description
Operating systems	C's portability and performance make it desirable for implementing operating systems, such as Linux and portions of Microsoft's Windows and Google's Android. Apple's OS X is built in Objective-C, which was derived from C. We discuss some key popular desktop/notebook operating systems and mobile operating systems in Section 1.11.
Embedded systems	The vast majority of the microprocessors produced each year are embedded in devices other than general-purpose computers. These **embedded systems** include navigation systems, smart home appliances, home security systems, smartphones, tablets, robots, intelligent traffic intersections and more. C is one of the most popular programming languages for developing embedded systems, which typically need to run as fast as possible and conserve memory. For example, a car's antilock brakes must respond immediately to slow or stop the car without skidding; game controllers used for video games should respond instantaneously to prevent any lag between the controller and the action in the game, and to ensure smooth animations.

Fig. 1.4 | Some popular performance-oriented C applications. (Part 1 of 2.)

Application	Description
Real-time systems	Real-time systems are often used for "mission-critical" applications that require nearly instantaneous and predictable response times. Real-time systems need to work continuously—for example, an air-traffic-control system must constantly monitor the positions and velocities of the planes and report that information to air-traffic controllers without delay so that they can alert the planes to change course if there's a possibility of a collision.
Communications systems	Communications systems need to route massive amounts of data to their destinations quickly to ensure that things such as audio and video are delivered smoothly and without delay.

Fig. 1.4 | Some popular performance-oriented C applications. (Part 2 of 2.)

Why not *C*?

- *C* is different !
  - » Data structures must be programmed “by hand”
  - » Operations must be done out in “long hand”
  - » No support for “object oriented” design
  - » Marginal support for higher-level thought processes
  - » *Much, much harder to use than higher level languages/systems*

- Better alternatives available for almost all applications
  - » *Java, Python, Ruby, etc.* – many CS situations
  - » *Matlab, SimuLink* – physical modeling
  - » *LabView* – instrumentation and control
  - » *Excel* – accounting and statistics
  - » *SQL* – billing and transactions

Popular C-based programming languages

Programming language	Description
Objective-C	Objective-C is an object-oriented language based on C. It was developed in the early 1980s and later acquired by NeXT, which in turn was acquired by Apple. It has become the key programming language for the OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).
Java	Sun Microsystems in 1991 funded an internal corporate research project which resulted in the C++-based object-oriented programming language called Java. A key goal of Java is to enable the writing of programs that will run on a broad variety of computer systems and computer-controlled devices. This is sometimes called "write once, run anywhere." Java is used to develop large-scale enterprise applications, to enhance the functionality of web servers (the computers that provide the content we see in our web browsers), to provide applications for consumer devices (smartphones, television set-top boxes and more) and for many other purposes. Java is also the language of Android app development.
C#	Microsoft's three primary object-oriented programming languages are Visual Basic (based on the original Basic), Visual C++ (based on C++) and Visual C# (based on C++ and Java, and developed for integrating the Internet and the web into computer applications). Non-Microsoft versions of C# are also available.

Fig. 1.5 | Popular C-based programming languages. (Part 1 of 3.)

Programming language	Description
PHP	PHP, an object-oriented, open-source scripting language supported by a community of users and developers, is used by millions of websites. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems. PHP also supports many databases, including the popular open-source MySQL.
Python	Python, another object-oriented scripting language, was released publicly in 1991. Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam (CWI), Python draws heavily from Modula-3—a systems programming language. Python is "extensible"—it can be extended through classes and programming interfaces.
JavaScript	JavaScript is the most widely used scripting language. It's primarily used to add dynamic behavior to web pages—for example, animations and improved interactivity with the user. It's provided with all major web browsers.

Fig. 1.5 | Popular C-based programming languages. (Part 2 of 3.)

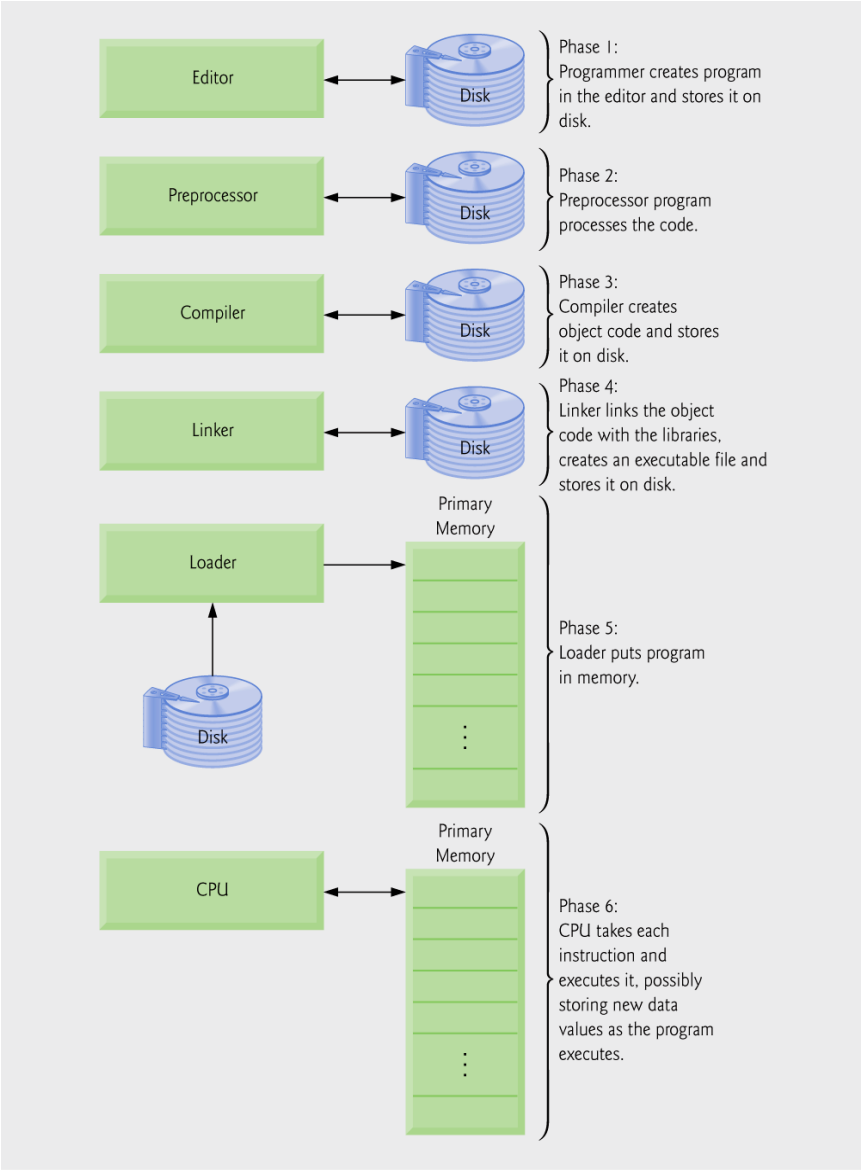
Programming language	Description
Swift	Swift, Apple's new programming language for developing iOS and Mac apps, was announced at the Apple World Wide Developer Conference (WWDC) in June 2014. Although apps can still be developed and maintained with Objective-C, Swift is Apple's app-development language of the future. It's a modern language that eliminates some of the complexity of Objective-C, making it easier for beginners and those transitioning from other high-level languages such as Java, C#, C++ and C. Swift emphasizes performance and security, and has full access to the iOS and Mac programming capabilities.

Fig. 1.5 | Popular C-based programming languages. (Part 3 of 3.)

C Program Development Environment

Standard Steps

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Typical C development environment.

Phase 1: Creating a Program

- Phase 1 consists of editing a file.
- This is accomplished with an editor program.
- Two editors widely used on Linux systems are vi and emacs.
- Software packages for the C/C++ integrated program development environments such as Eclipse and Microsoft Visual Studio have editors that are integrated into the programming environment.
- You type a C program with the editor, make corrections if necessary, then store the program on a secondary storage device such as a hard disk.
- C program file names should end with the .c extension.

Phases 2 and 3: Preprocessing and Compiling a C Program

- In Phase 2, the you give the command to compile the program.
- The compiler translates the C program into machine language-code (also referred to as object code).
- In a C system, a preprocessor program executes automatically before the compiler's translation phase begins.
- The C preprocessor obeys special commands called preprocessor directives, which indicate that certain manipulations are to be performed on the program before compilation.

Phases 2 and 3: Preprocessing and Compiling a C Program (Cont.)

- These manipulations usually consist of including other files in the file to be compiled and performing various text replacements.
- The most common preprocessor directives are discussed in the early chapters; a detailed discussion of preprocessor features appears in Chapter 13.
- In Phase 3, the compiler translates the C program into machine-language code.
- A syntax error occurs when the compiler cannot recognize a statement because it violates the rules of the language.
- Syntax errors are also called compile errors, or compile-time errors.

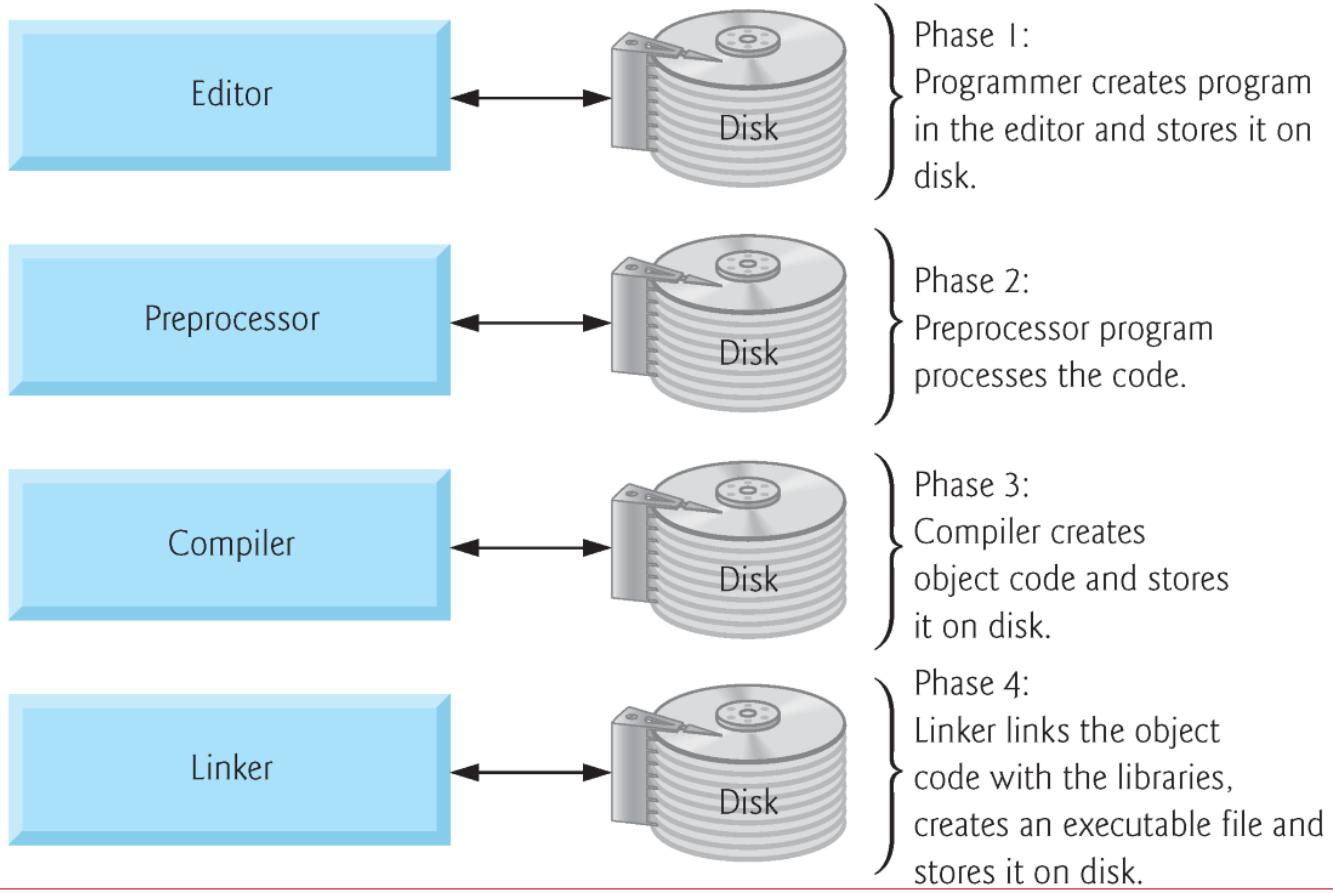


Fig. 1.6 | Typical C development environment. (Part 1 of 2.)

Phase 4: Linking

- The next phase is called linking.
- C programs typically contain references to functions defined elsewhere, such as in the standard libraries or in the private libraries of groups of programmers working on a particular project.
- The object code produced by the C compiler typically contains “holes” due to these missing parts.
- A linker links the object code with the code for the missing functions to produce an executable image (with no missing pieces).
- On a typical Linux system, the command to compile and link a program is called gcc (the GNU compiler).

Phase 4: Linking (Cont.)

- To compile and link a program named `welcome.c` type

  `» gcc welcome.c`

- at the Linux prompt and press the *Enter* key (or *Return* key).
- [Note: Linux commands are case sensitive; make sure that each c is lowercase and that the letters in the filename are in the appropriate case.]
- If the program compiles and links correctly, a file called `a.out` is produced.
- This is the executable image of our `welcome.c` program.

Phase 5: Loading

- The next phase is called loading.
- Before a program can be executed, the program must first be placed in memory.
- This is done by the loader, which takes the executable image from disk and transfers it to memory.
- Additional components from shared libraries that support the program are also loaded.

Phase 6: Execution

- Finally, the computer, under the control of its CPU, executes the program one instruction at a time.

- To load and execute the program on a Linux system, type `./a.out` at the Linux prompt and press *Enter.*

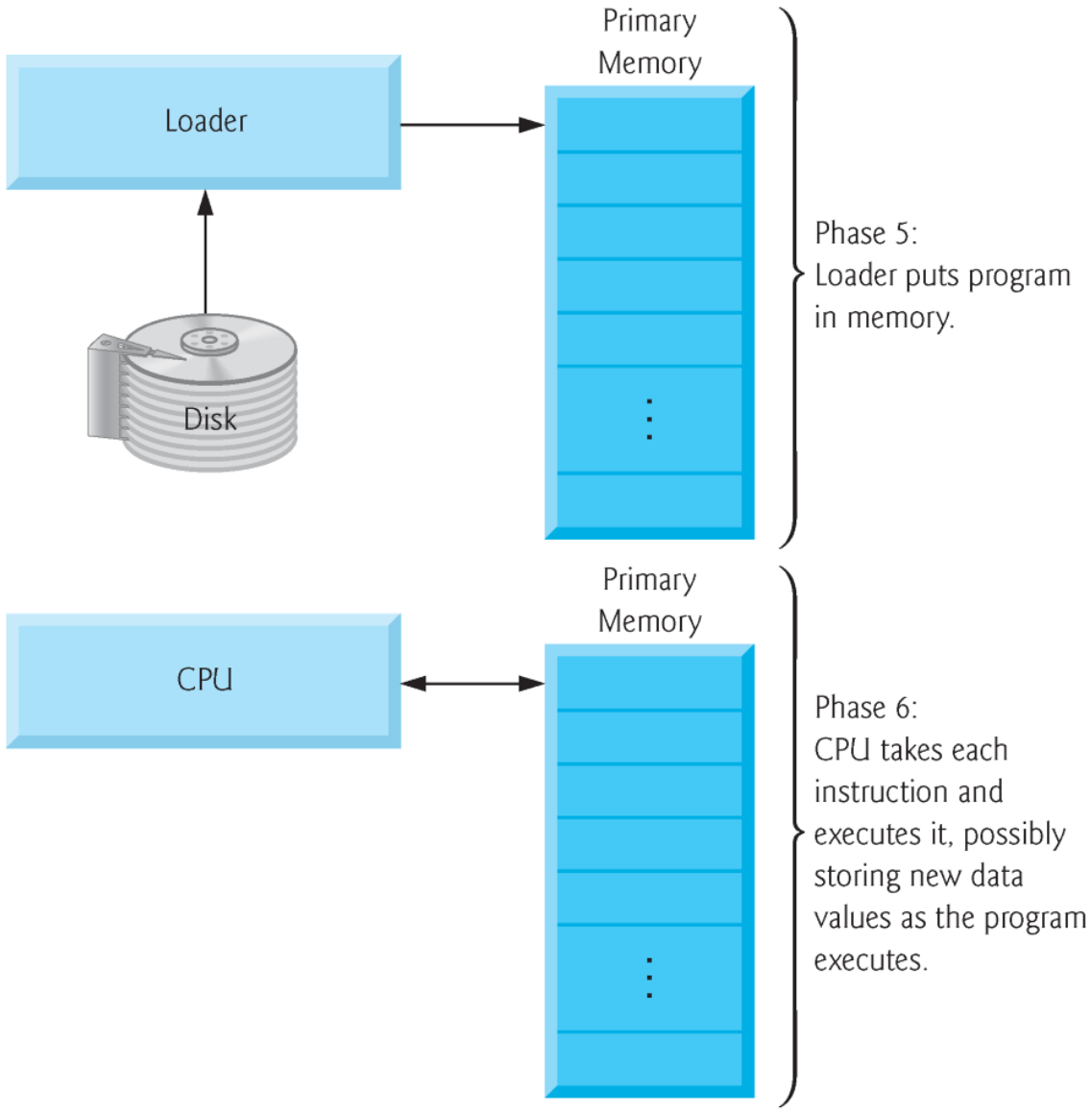


Fig. 1.6 | Typical C development environment. (Part 2 of 2.)

Problems That May Occur at Execution Time

- Programs do not always work on the first try.
- Each of the preceding phases can fail because of various errors that we'll discuss.
- For example, an executing program might attempt to divide by zero (an illegal operation on computers just as in arithmetic).
- This would cause the computer to display an error message.
- You would then return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine that the corrections work properly.

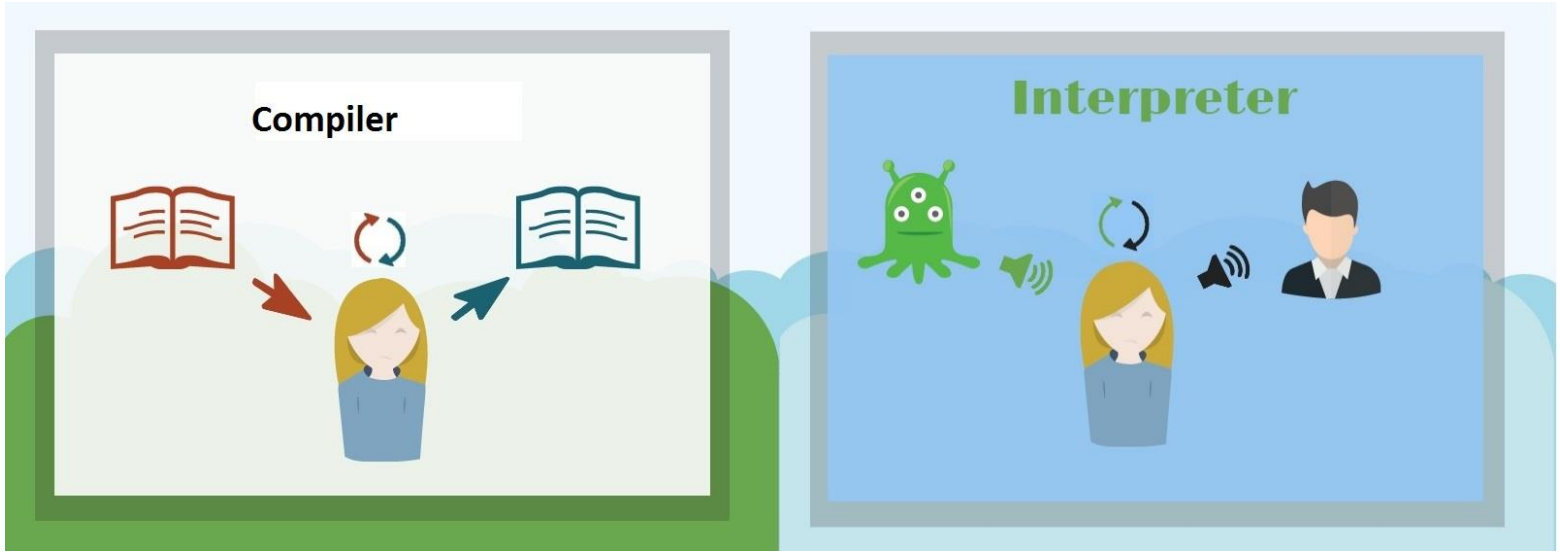
Machine Languages, Assembly Languages and High-Level Languages

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps.
- Any computer can directly understand only its own machine language, defined by its hardware design.
- Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.
- Programming in machine language—the numbers that computers could directly understand—was simply too slow and tedious for most programmers.
- Instead, they began using English like abbreviations to represent elementary operations.
- These abbreviations formed the basis of assembly languages.
- *Translator programs* called assemblers were developed to convert assembly-language programs to machine language.

Machine Languages, Assembly Languages and High-Level Languages

- Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language.
- To speed the programming process even further, high-level languages were developed in which single statements could be written to accomplish substantial tasks.
- High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical expressions.
- Translator programs called compilers convert high-level language programs into machine language.
- Interpreter programs were developed to execute high-level language programs directly, although more slowly than compiled programs.
- Scripting languages such as JavaScript and PHP are processed by interpreters.

Compiler vs Interpreter



Compiler vs Interpreter

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Ruby PHP JAVA Perl R Powershell	Programming language like C C++ C# Objective-C SWIFT Fortran.