EXPERT CONTRIBUTORS +4

Compiler vs. Interpreter in Programming

Compilers and interpreters are used to convert a high-level language into machine code. However, there are differences between how and when they work.

Written by Rakia Ben Sassi

Published on Apr. 24, 2023

Image: Shutterstock / Built In

I still remember a discussion with a colleague in which I said, “That’s the transpiler,” and he replied, “The…what?”

To speak to a computer in its non-human language, we came up with two solutions: interpreters and compilers. Ironically, most of us know very little about them, although they're a part of our daily coding life.

COMPILERS VS. INTERPRETERS EXPLAINED

- Compiler: A compiler translates code from a high-level programming language (like Python, JavaScript or Go) into machine code before the program runs.
- Interpreter: An interpreter translates code written in a high-level programming language into machine code line-by-line as the code runs.

In this post, I'll dive into the journey of translating a high-level language into a machine code ready for execution. I'll focus on the inner workings of the two key players in this game, the compiler and the interpreter, and break down the related concepts.

Compilers vs. Interpreters: How Do They Work?

Compilers and interpreters have long been used as computer programs to transform code. But they work in different ways:

- A compiler translates code written in a high-level programming language into a lower-level language like assembly language, object code and machine code (binary 1 and 0 bits). It converts the code ahead of time before the program runs.

Compilers vs. Interpreters: Advantages and Disadvantages

Both compilers and interpreters have pros and cons:

- A compiler takes in the entire program and requires a lot of time to analyze the source code. Whereas the interpreter takes a single line of code and very little time to analyze it.
- Compiled code runs faster, while interpreted code runs slower.
- A compiler displays all errors after compilation. If your code has mistakes, it will not compile. But the interpreter displays errors of each line one by one.
- Interpretation does not replace compilation completely.
- Compilers can contain interpreters for optimization reasons like faster performance and smaller memory footprint.

A high-level programming language is usually referred to as “compiled language” or “interpreted language.” However, in practice, they can have both compiled and interpreted implementations. C, for example, is called a compiled language, despite the existence of C interpreters. The first JavaScript engines were simple interpreters, but all modern engines use just-in-time (JIT) compilation for performance reasons.

4 Common Types of Interpreters to Know

Interpreters were used as early as 1952 to ease programming and also translate

4 TYPES OF INTERPRETERS

1. Bytecode interpreter
2. Threaded code interpreter
3. Abstract syntax tree interpreter
4. Justin-in-time compilation

1. BYTECODE INTERPRETER

The trend toward bytecode interpretation and just-in-time compilation blurs the distinction between compilers and interpreters. Bytecode interpreters can process up to 256 instructions, with each instruction starting with a byte.

2. THREADED CODE INTERPRETER

Unlike bytecode interpreters, threaded code interpreters use pointers instead of bytes. Each instruction is a word pointing to a function or an instruction sequence, possibly followed by a parameter. The number of different instructions is limited by the available memory and address space.

Forth code, which is used in Open Firmware systems, is a classical example of threaded code. The source code is compiled into a bytecode known as “F code,” which a virtual machine then interprets.

3. ABSTRACT SYNTAX TREE INTERPRETER

native code just-in-time.

AST keeps the global program structure and relations between statements. This allows the system to perform better analysis during runtime and makes AST a better intermediate format for just-in-time compilers than bytecode representation.

However, for interpreters, AST causes more overhead. Interpreters walking the abstract syntax tree are slower than those generating bytecode.

4. JUST-IN-TIME COMPILATION

Just-in-time compilation (JIT) is a technique in which the intermediate representation is compiled to native machine code at runtime.

10 Common Types of Compilers to Know

Below are the common types of compilers you should know.

10 TYPES OF COMPILERS

1. Cross-compiler
2. Native compiler
3. Bootstrap compiler
4. Decompiler
5. Source-to-source compiler
6. Language rewriter

1. CROSS-COMPILER

A cross-compiler is one that runs on a computer whose CPU or operating system differs from the one on which the code it produces will run.

2. NATIVE COMPILER

A native compiler produces an output that would run on the same type of computer and operating system as the compiler itself.

3. BOOTSTRAP COMPILER

Bootstrap compiler is a compiler written in the language that it intends to compile.

4. DECOMPILER

A decompiler translates code from a low-level language to a higher level one.

5. SOURCE-TO-SOURCE COMPILER (TRANSPILER)

A source-to-source compiler is a program that translates between high-level languages. This type of compiler is also known as a transcompiler or transpiler.

Some examples of a transpiler include:

intermediate code, since the generated code was not usually intended to be readable by humans.

6. A LANGUAGE REWRITER

This is usually a program translating form of expressions without a change of language.

7. BYTECODE COMPILER

A compiler that translates a high-level language into an intermediate simple language that a bytecode interpreter or virtual machine can interpret. Examples include: Bytecode compilers for Java and Python.

8. JUST-IN-TIME COMPILER (JIT COMPILER)

A JIT compiler defers compilation until runtime. It generally runs inside an interpreter.

Examples of a JIT compiler include:

- The earliest published JIT compiler is attributed to LISP in 1960.
- The latter technique appeared in languages such as Smalltalk in the 1980s.
- Since then, JIT compilation has gained mainstream attention amongst modern languages like Java, .NET Framework, Python and most modern

executes the bytecode, and later the JIT compiler translates the bytecode to machine code.

Java bytecode can either be interpreted at runtime by a virtual machine, or compiled at load time or runtime into native code. Modern JVM implementations use the compilation approach, so after the initial startup time the performance is equivalent to native code.

9. AOT COMPILATION

Ahead-of-time (AOT) compilation is the approach of compiling a higher-level programming language, or an intermediate representation such as Java bytecode, before the runtime.

An example of this is the Angular framework. This uses an ahead-of-time (AOT) compiler to transform HTML and TypeScript code into JavaScript code during the build time to provide a faster rendering later on the browser when the code is running.

10. ASSEMBLER

An assembler translates human-readable assembly language into machine code. This compilation process is called assembly. The inverse program that converts machine code to assembly language is called a disassembler.

An assembly language (ASM) is a low-level programming language in which there is a dependence on the machine code instructions. That's why every assembly language is designed for exactly one specific computer architecture.

A tutorial on the differences between compilers and interpreters. | Video: ISO Training Institute

MORE ON SOFTWARE DEVELOPMENT
What Is the Java Runtime Environment?

Why Compilers and Interpreters Are Important

Both compilers and interpreters are computer programs that convert a code written in a high-level language into a lower-level or machine code understood by computers. However, there are differences in how they work and when to use them.

Even if you're not going to implement the next compiler or interpreter, these insights should help to improve your knowledge of the tools you use as a developer every day.

RECENT EXPERT CONTRIBUTORS ARTICLES

How to Use JSON Schema to Validate JSON Documents in Python

Why Psychological Safety Is the Key to a Great Product Team

Will Twitter's New Rate Limits Really Stop Scraping?

Expert Contributors Javascript Python

FOR EMPLOYERS

Get Involved

Recruit With Built In

Become an Expert Contributor

Send Us a News Tip

Resources

Customer Support

Share Feedback

Report a Bug

Tech A-Z

Browse Jobs

Tech Hubs

Built In Austin

Built In Boston

Built In Chicago

Built In Colorado

Built In LA

Built In NYC

Built In San Francisco

Built In Seattle

See All Tech Hubs