How does an interpreter/compiler work

Ask Question

How does an interpreter/compiler work? What is the difference between interpreter and compiler.

compiler-construction interpreter

asked Mar 4 '10 at 6:24
developer
1,460 4 22 34

You can lookup - Difference between Compiler, Interpreter and Assembler – Aniket Thakur May 14 '17 at 7:58

8 Answers

Compilers

Compilers were the first sort of translator program to be written. The idea is simple: You write the program, then hand it to the compiler which translates it. Then you run the result.

Interpreters

An interpreter is also a program that translates a high-level language into a low-level one, but it does it at the moment the program is run. You write the program using a text editor or something similar, and then instruct the interpreter to run the program. It takes the program, one line at a time, and translates each line before running it: It translates the first line and runs it, then translates the second line and runs it etc.

Compiler characteristics:

- spends a lot of time analyzing and processing the program
- the resulting executable is some form of machine- specific binary code
- the computer hardware interprets (executes) the resulting code
- program execution is fast

Interpreter characteristics:

- relatively little time is spent analyzing and processing the program
- the resulting code is some sort of intermediate code
- the resulting code is interpreted by another program
- program execution is relatively slow

edited Jun 11 '17 at 0:25
EJP
248k 22 194 330

answered Mar 4 '10 at 6:29
Adriaan Stander
125k 14 226 248

so c# uses both compiler and interpreter? – developer Mar 4 '10 at 6:49

1 @user283405: VB is interpreted, while C# is compiled. – KMån Mar 4 '10 at 10:02

8 That's not correct. Both languages are both compiled and interpreted. – Matthew H May 22 '10 at 23:25

@Matt H: Yep, that would be correct if you are counting the runtime as well; because runtime interprets the complied assemblies. From C#'s point of view, it is first compiled and then executed. – KMån Jan 3 '12 at 9:28

1 @Aniket nope, no interpretation takes place on object code. If you compile a program into something the machine understands, then you dont need an interpreter. Moreover, the point of the question is the difference between the object code produced by a compiler and the midcode (eg: bytecode) produced for example by a Java compiler that is then interpreted by a real Interpreter each time its run. – Juan Jul 7 '14 at 12:06

|

What is a translator?

> An **S -> T translator** accepts code expressed in source language S, and translates it to equivalent code expressed in another (target) language T.

Examples of translators:

- Compilers - translates high level code to low level code, e.g. *Java -> JVM*
- Assemblers - translates assembly language code to machine code, e.g. *x86as -> x86*
- High-level translators - translates code from one PL to another, e.g. *Java -> C*
- Decompilers - translates low-level code to high-level code, e.g. *Java JVM bytecode -> Java*

What is an interpreter?

> An **S interpreter** accepts code expressed in language S, and immediately executes that code. It works by fetching, analysing, and executing one instruction at a time.

Great when user is entering instructions interactively (think Python) and would like to get the output before putting in the next instruction. Also useful when the program is to be executed only once or requires to be portable.

- Interpreting a program is much slower than executing native machine code
- Interpreting a high-level language is ~100 times slower
- Interpreting an intermediate-level (such as JVM bytecode) language is ~10 slower
- If an instruction is called repeatedly, it will be analysed repeatedly - time-consuming!
- No need to compile code

Differences

Behaviour

- A compiler translates source code to machine code, but does not execute the source or object code.
- An interpreter executes source code one instruction at a time, but does not translate the source code.

Performance

- A compiler takes quite a long time to translate the source program to native machine code, but subsequent execution is fast
- An interpreter starts executing the source program immediately, but execution is slow

Interpretive compilers

An interpretive compiler is a good compromise between compilers and interpreters. It translates source program into virtual machine code, which is then interpreted.

An interpretive compiler combines fast translation with moderately fast execution, provided that:

- VM code is lower than the source language, but higher than native machine code
- VM instructions have simple formats (can be quickly analysed by an interpreter)

Example: JDK provides an interpretive compiler for Java.

edited Oct 27 '15 at 5:18
EJP
248k 22 194 330

answered May 4 '14 at 12:48
martynas
8,834 3 38 53

3 "An interpreter executes source code one instruction at a time,but does not translate the source code"(taken from diff heading) I think interpreter translates and executes both actions at a time but not just executing as you explained. apart from this explanation was clear and perfect +1. – JAVA Jul 1 '14 at 16:40

Compiler, transforms source code in one computer language to another one.

Interpreter, executes source code directly (usually inside its own virtual machine).

alt text http://content.answers.com/main/content/img/CDE/COMPILE.GIF

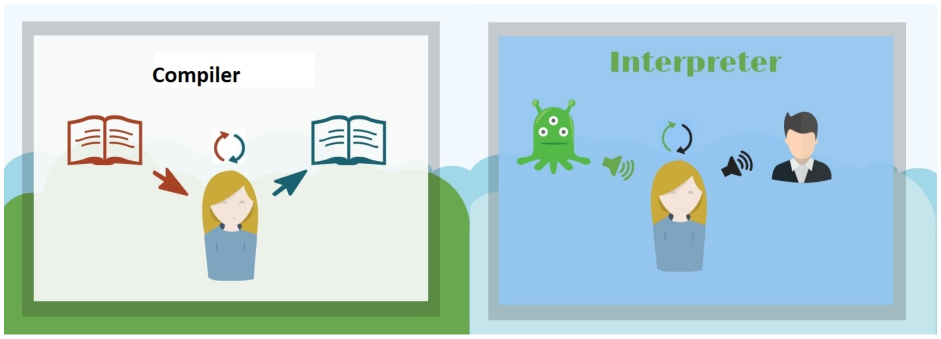
Generally interpreter is performance costly.

answered Mar 4 '10 at 7:31
KMån

9,225 2 24 38

Nearly all interpreters execute compiled byte code, and some (simulators) execute compiled machine code. Interpretation is only 'performance costly' if the instructions being interpreted are simple compared to the fetch overhead. – EJP Jun 11 '17 at 0:22

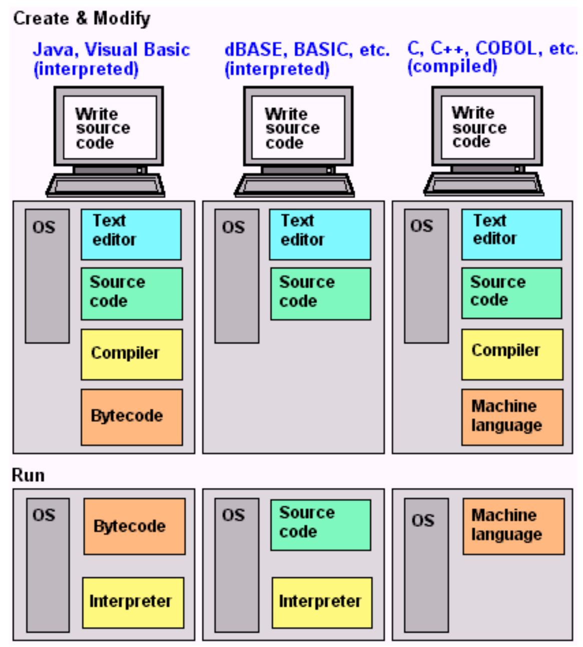
> What is the difference between interpreter and compiler?



The **Compiler** translates the entire program before it is run.

The **Interpreters** translates one statement into machine language, executes it, and proceeds to next statement.

> How does an interpreter/compiler work?



- Unlike compiled languages which are translated into machine language ahead of time (right).
- Interpreted languages are translated at runtime.
- dBASE and BASIC **interpreters** (middle) translate the original source code.
- Java and Visual Basic (left) **interpreters** translate **bytecode**, which is an intermediate language compiled from the original **source code**.

Source

edited Sep 8 '17 at 16:36

answered Jul 21 '15 at 23:18
Premraj
21.2k 9 118 98

'A line at a time' is not correct. Most interpreters precompile the code to a p-code and then execute that one p-code instruction at a time. – EJP Oct 27 '15 at 5:19

Interpreters do not 'translate byte code'. They *execute* it. There are Basic implementations that execute byte code, not source code. – EJP Jun 11 '17 at 0:47

The Difference vs How they Work

Q: *What are Compilers and interpreters used for ?*

A: Most programs are written in High-level languages (c#,java...). High-level languages are which contain understandable words and phrases. On the other hand computer (by the time i wrote this article that is) understand machine code which is 0's and 1's only aka Binary/Machine Code. Therefore we need to convert the High-level code into source code which is (Machine code/binary). Hence the word converting.

So we conclude a Compiler/interpreter job is to Translate high-level code into machine code.

But both have a different way of 'Translating' the code **Difference :**

Compiler:

converts source code to some kind of intermediate form. For static language, a compiler usually converts the source code to assembly, which usually did not get stored to disk, then the assembler is invoked to convert the assembly to binary code, which is usually stored as object file(.o or .obj suffix usually), then linker is invoked to link object file(s) to binary executable. Also it is common to refer to this whole process of compiling, assembling, linking as compiling. So you may call gcc a compiler, but it actually invokes cc1 which is the compiler to compile, as which is the assembler to assemble, ld which is the linker to link.

Interpreters : language which has a intermediate so called bytecode form, the source code is first converted to byte code, this process can be called compiling. Bytecode cannot be run on host machines, it needs a program, which is actually the process from the viewpoint of OS, to interpret the bytecode to the host machine, this program is called a interpreter, think of java. Some language, like python, do the compiling and interpreting work with a single.

Comparing

Interpreters

- It takes less amount of time to analyze the source code but the overall execution time is slower.
- No intermediate object code is generated, hence are **memory** efficient.
- Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.

Compilers

- It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
- Generates intermediate object code which further requires linking, hence requires more memory.
- It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.

Examples with Languages

Interpreted

- Python
- Ruby
- PHP
- JAVA(Almighty)
- Perl
- R
- Powershell

compiled

- C
- C++
- C#
- Objective-C
- SWIFT
- Fortran

answered Jun 5 '15 at 14:31 community wiki
Lowkey papi

Most compilers do not generate assembly language. Linking does not imply more memory. Source code interpreters exist. C# is interpreted, or indeed Java and C# are both compiled and interpreted.. C and C++ exist in interpreted versions, as does Fortran. – EJP Jun 11 '17 at 0:44

Compilers - A compiler translates a source language to a target language. Then the target language takes input and gives output.

Compiler produces a target code by compiling source code

then target code then takes input and gives output

Interpreters - Instead of generating a target code, Interpreter appears to take input directly along with source code and gives output.

Interpreter maps input to output using the source program

Then machine-language targeted program produced by the compiler is much faster than the interpreter at mapping inputs to output. However, since interpreter executes source program line by line it gives better error diagnostics than compiler.

Reference - Compilers: Principles, Techniques, and Tools by Aho aka the dragon book

answered Feb 25 at 9:21
vaibhaw.vipul
20 5

Compiler - A compiler is a computer program that transforms (translates) source code of a programming language (the source language) into another computer language (the target language). In most cases compilers are used to transform source code into executable program, i.e. they translate code from high-level programming languages into low (or lower) level languages, mostly assembly or machine code.

Interpreter - An interpreter is a computer program that executes instructions written in a programming language. It can either execute the source code directly or translates the source code in a first step into a more efficient representation and executes this code

answered Oct 27 '15 at 5:02
Krishna
2,037 1 18 24

Look at **PLAI** book, it's the best intro to dymanic language realization I found ever:

Programming Languages: Application and Interpretation (c) *Shriram Krishnamurthi*

- [home] https://cs.brown.edu/~sk/Publications/Books/ProgLangs/2007-04-26/
- [html] http://cs.brown.edu/courses/cs173/2012/book/
- [pdf] http://www.cs.brown.edu/courses/cs173/2012/book/book.pdf

This book centers on writing interpreter for dynamic language in Scheme (dr.Racket), using it you can write your own interpreter for any language, and add some tips on OOP from

- [OOP PLAI] https://users.dcc.uchile.cl/~etanter/ooplai/ (c) Éric Tanter

and SmallTalk and SOM: Simple Object Machine:

- Virtual Machines for Teaching and Research
- A minimal Smalltalk for teaching of and research on Virtual Machines
- Smalltalk-80: The Language and its Implementation

All modern interpreters include compiler inside: compile highlevel elements **into** low-level but portable **byte-code**, or use JIT for compiling into machine code into RAM.

PS: If anybody wants to write SmallTalk system on Python, please note me.

answered Mar 28 '17 at 7:17
Dmitry Ponyatov
172 11

1 90% of this answer is just advertising. – EJP Jun 11 '17 at 0:27