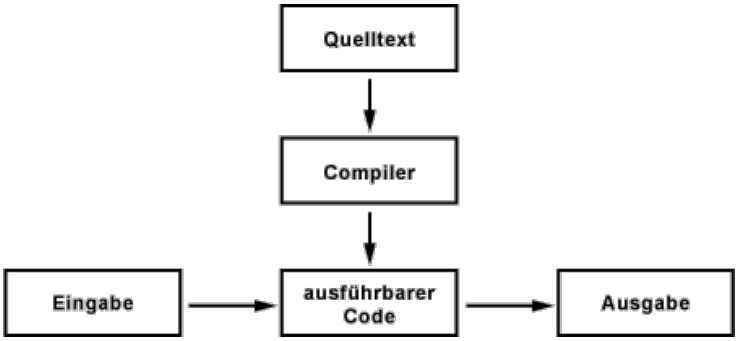
Compiler und Interpreter

Compiler und Interpreter sind Implementierungsformen von Software. Generell geht es beim Compilieren und Interpretieren darum, den Quelltext, der mit einer höheren Programmiersprache (zum Beispiel C++, C# oder Java) geschrieben wurde, in Maschinenbefehle umzusetzen. Das bedeutet, die lesbaren Programmierbefehle müssen in weniger komplexe Instruktionen übersetzt werden, damit der Prozessor diese ausführen kann.
In der Theorie kann der Quelltext jeder Programmiersprache sowohl mit einem Compiler als auch mit einem Interpreter implementiert werden. In der Praxis ist die Implementierung trotzdem festgelegt. Die Verwendung einer Programmiersprache legt den Einsatz eines Compilers oder Interpreters fest.

Compiler



Der Compiler ist ein Programm, der aus dem Quelltext das eigentliche Programm erstellt. Der Quelltext könnte zum Beispiel in C oder C++ geschrieben sein. Einzelne Anweisungen aus dem Quelltext werden in Folgen von Maschinenanweisungen übersetzt. Nach dem Compilieren wird das Programm erzeugt, in dem sich ausführbarer Code befindet. Bei der Ausführung des Programms werden diese Anweisungen "direkt" vom Prozessor ausgeführt. Jede Maschinenanweisung entspricht dabei einer festgelegten Bitfolge. Innerhalb des Prozessors wird diese Bitfolge verarbeitet. Der ausführbare Code verarbeitet auch Eingaben und erzeugt die Ausgabe.

Beim Compilieren von Software, legt der Compiler fest, welche Instruktionen in welcher Reihenfolge an den Prozessor geschickt werden. Wenn diese Instruktionen nicht auf andere warten müssen, kann der Prozessor sogar mehrere davon parallel verarbeiten.
Sobald ein neues Betriebssystem oder ein neuer Prozessor zum Einsatz kommt, muss der Quelltext neu compiliert werden. In der Praxis ist es so, dass Prozessoren und Betriebssysteme einen Kompatibilitätsmodus haben, so das alte Programme auch auf neuen Plattformen laufen. So müssen compilierte Programme nicht immer neu compiliert werden. Aber die Kompatibilität hat natürlich ihre Grenzen.

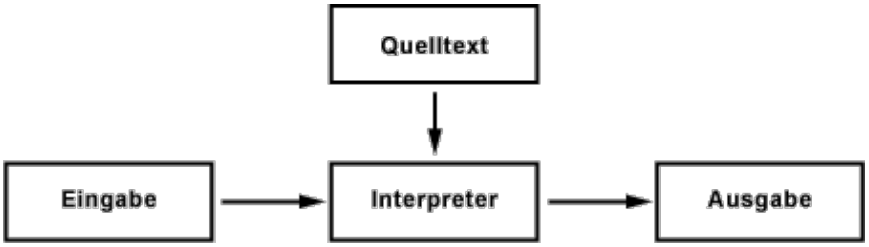
Für jede Programmiersprache (m) braucht es für jeden Prozessor (n) einen eigenen Compiler (m x n). Typische Programmiersprachen mit Compiler sind Pascal, Modula, COBOL, Fortran, C und C++.

Vorteile: Die Übersetzung in ausführbaren Code ist äußerst effizient und optimiert den generierten Code. Compilierte Programme arbeiten sehr schnell, was sich besonders bei lang laufenden Programmen lohnt.

Nachteile: Der Aufwand bei der Software-Entwicklung steigt durch das Compilieren, was einiges an Zeit und Ressourcen in Anspruch nimmt. So muss bei jeder Quelltext-Änderung erneut compiliert werden, wenn das Programm getestet werden soll.

Interpreter

Der Quelltext wird von der Programmiersprache in einen Hardware-unabhängigen Bytecode umgewandelt. Der Interpreter setzt diesen Bytecode in Maschinenanweisungen um. Den Interpreter muss man sich dabei als eine virtuelle Maschine (VM) vorstellen, in der das Programm läuft. Hierbei darf man den Interpreter nicht mit einer virtuellen Maschine der Virtualisierungstechnik verwechseln. Es handelt sich dabei um zwei unterschiedliche Techniken mit der gleichen Bezeichnung.



Der Interpreter ist eine Software-Bibliothek, die Eingaben und Bytecode entgegennimmt und zur Laufzeit interpretiert. Der Interpreter agiert als eine virtuelle Maschine, die den Bytecode ausführt. Der Interpreter nimmt dabei die Rolle des Prozessors ein.

Die virtuelle Maschine ist als eine Zwischenschicht zwischen Programmiersprache und Prozessor zu verstehen. Es handelt sich sich um einen Software-Container. Jetzt braucht man nur noch einen Interpreter pro Prozessor.
Typische Programmiersprachen mit Interpreter sind BASIC, Smalltalk, LISP und Python.

Vorteile: Bei der Entwicklung der Software kann man sofort testen, was das Debugging (Fehlersuche) erleichtert. Der verwendete ausführbare Code wird erst zur Laufzeit generiert.

Nachteile: Generell sind interpretierte Programme langsamer und ineffizient. Es müssen immer die selben Programmteile, wie zum Beispiel Schleifen und Funktionen, erneut übersetzt werden.

JIT-Compiler (JIT = Just in time)

Es gibt auch Programmiersprachen, wie zum Beispiel C# und Java, die Compiler und Interpreter kombinieren. Wie beim Interpreter wird der Quelltext in Hardware-unabhängige VM-Anweisungen (Bytecode) übersetzt. Bei der Ausführung dieser Anweisungen durch den JIT-Compiler werden die Anweisungen in Maschinencode übersetzt. Der JIT-Compiler berücksichtigt dabei die verschiedenen Eigenheiten des Prozessors.

Beim JIT-Compiler wird aus dem Bytecode der Programmiersprache der Maschinencode für den Prozessor generiert, weil das schneller ist, als wenn der Interpreter den Bytecode ausführt. Der Bytecode wird nach der Ausführung verworfen und muss bei nochmaliger Ausführung erneut übersetzt werden.

Typischerweise arbeitet man mit einem JIT-Compiler, wenn es um das Prinzip "write once, run anywhere" geht.
Wenn es also darum geht, den Quelltext auf unterschiedlichen Betriebssystemen und Hardware einzusetzen.
Eine Software wird einmal geschrieben und ist dann auf verschiedenen Plattformen lauffähig (Portabilität).