Compilers and Interpreters

Bhavneet Soni

Author's Note

Computers are used for a vast number of applications; this has led to development of different computer architectures and programming languages. Since humans and computers think in different languages, hence the need for a translator that could convert human understandable language into computer understandable code. Compilers and interpreters are set of programs that could effectively communicate with the machine, they are imperative to harness the full computing prowess. I have discussed the key differences between compilers and interpreters, and explored advantages and disadvantages of different types of compilers.

Compilers and Interpreters

Unlike humans, computers can understand only one language, language of 0's and 1's. Where as humans can not think or understand in just those terms, they can communicate in human languages such as English, Spanish, French, Hindi etc. But these languages are too complex for computers to understand. To tell a computer what to do in human terms we use Progrmaming languges which are not plain human languages but still readable by humans have a regid predefined syntax that elaborates step by step instructions. Code written in programming languages is still not what a comptuer can interpret and hence the need to have a go between translator that would translate a human readable instruction set i.e source code into some thing that computer can interpret and execute. Compiler is a software that checks the sourse code and list the errors and or discrepencies which will stop the comptuer from following the instructions. Once the errors are fixed compiler outputs a binary code instruciton also known as machine code which a computer can understand and execute. Although primary purpose of compilers and interpreters is essentially the same, but there is a difference how they handle the whole code and the data inputs.

Compilers and Interpreters

Following are some of the key differnecs between a Compiler and an Interpreter (Kumar, n.d.)

1. Main difference between compiler and an interpreter is that compiler takes in the whole code, checks it for errors and outputs the binary code where as interpreter go line by line, it takes one statement at a time and executes it.

2. Interpreter will not know of an error untill it comes across the statement that has error and will execute the program uptill that time, however compiler will not run the program until all the errors are fixed.
3. Compiler generates intermediate code known as Object code or machine code whereas interpreter does not need to do this.

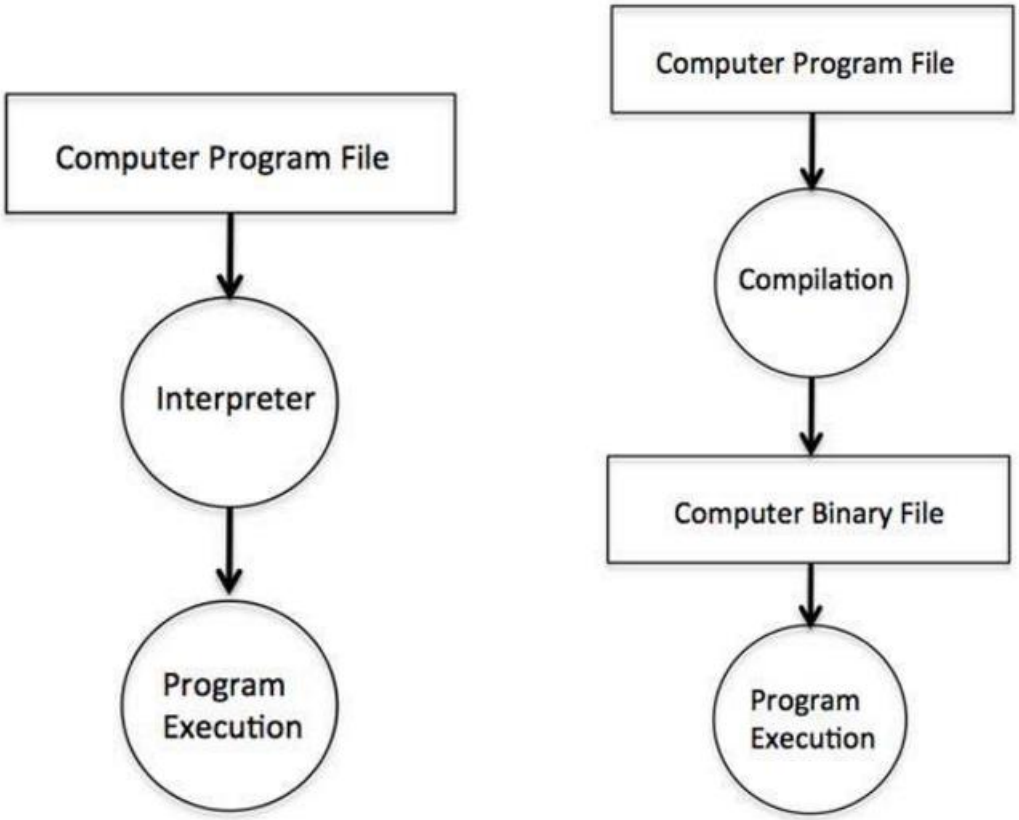

Figure 1- Steps in Compiler vs Interpreter (http://www.tutorialspoint.com, n.d.)

4. Conditional statements suchas if else are executed much faster by a compiler inlue of the interpreter.
5. Compilers takes up more memory as the complete object code resides in memory whereas interpreter is more efficient in terms of memory usage.
6. Error handling is better by compiler as entire source code is checked for errors and other sytacical errors and will not let the program run untill all the errors are fixed,

compiler are btter at error handling but not efficient in debugging the errors while interpreter stops at the first error it encounters and will not know of other errors until the first one is fixed making debugging easy.

7. Once a complier has generated the compiled code the program it can be stored in memory and can be run any number of times with different data inputs however this is not possible with the interpreted code.

Figure 1 shows a graphical representation of the complier and interpreter steps. Interpreters are more closely related to how a human brain thinks i.e sequentially hence interpreter languages are generally preferred for learning to code and for coding smaller applications. Examples of compiled languages are Java, C++, C#, COBOL etc, while languages like Visual Basic, Python, PHP, Ruby, MATLAB, Lisp, JavaScript Perl etc. are interpreter based. Figure 2 below show how different programing languages are converted from source code into machine code.

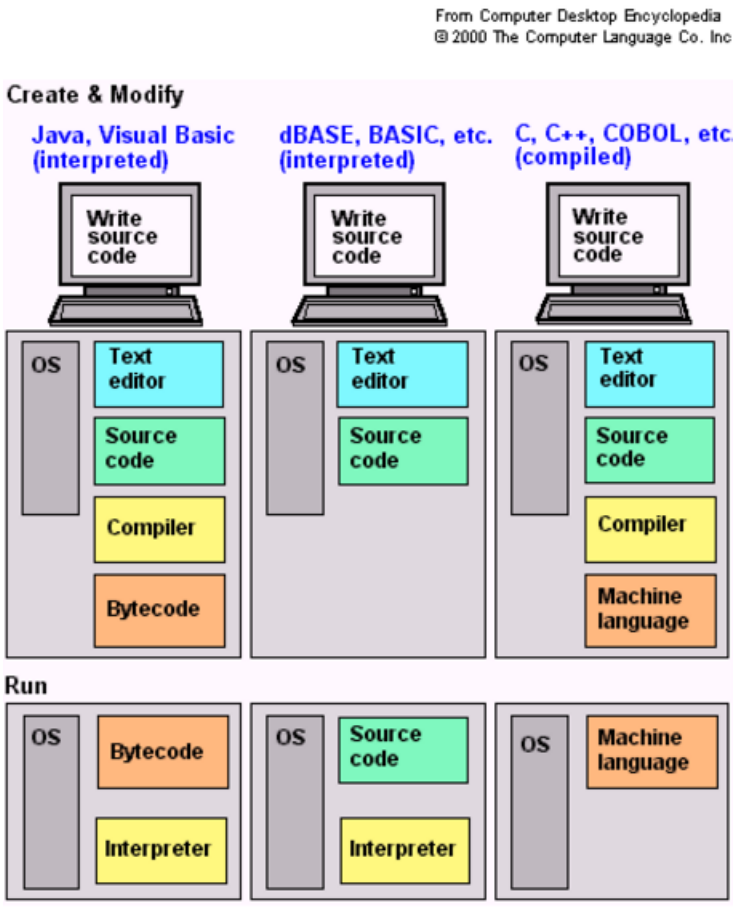

Figure 2- Compiler and Interpreters used by different programming languages (Kumar, n.d.)

Compilers and interpreters are not always exclusive one curious case is for Java, it is both a compiled and interpreted language. JAVAC compilers converts .java source code file to .class file Java interpreter converts .class file into machine-readable instructions written in java bytecode. This bytecode is loaded on a Java Virtual Machine (JVM) which can be run on any platform, JVM then interprets the bytecode file and converts it to binary code to be executed. Another popular complier is Unix g++, it is used to compile C++ source code and gives an output in a .out format which can be then can run on sparc microprocessor. Since Compilers and interpreters convert source code to machine understandable code they have to be aware of the computer architecture the program is running and hence these need to be specific for each

architecture. To allow for cross architecture certain interpreters are available such as VirtualPC interprets programs written for the Intel Pentium architecture (IBM-PC clone) for the PowerPC architecture (Macintosh). This enable Macintosh users to run Windows programs on their computer (Robert Sedgewick, 2008).

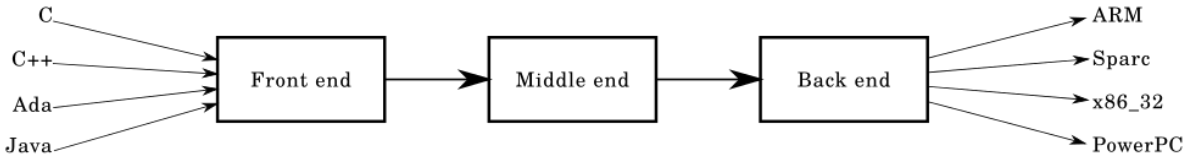

Figure 3- Compilers design (wikipedia.com, n.d.)

Classification of Compilers

Compilers essentially consists of 3 interfaces, Front, Middle and Back end. Front end verifies syntax and semantics as per the language conventions of the High Level language used to write the source code. Front end is responsible for throwing errors and warnings in the source code. Front end generates an intermediate representation (IR) that conforms to the lexical and semantic syntax.

The middle end performs takes in the IR and optimizes it by removing useless or unreachable code, finds and manages location of constant values, such as moving it out a less frequently executed place (e.g., out of a loop), or special manipulation based on the context. Middle end layers essentially are designed so be independent of the source code language and computer architecture the program will be running on, middle end thus generates another IR for to be used by the back end.

Back end takes in the IR generated by middle end analyze transform and optimize for a particular computer architecture. It generates an assembly code that is dependent on the computer architecture and performs register allocation in process. Backend is responsible for

optimizing the code utilization of the hardware, and harness the power of the computer. Backend generates machine code that is specialized for the processor and operating system of the machine. Modern day compliers vary on how many steps it takes to convert the source code in to the machine-readable code, based on the number of steps compilers are classified as

1. Single pass compilers (one-pass)
2. Multi pass compilers
3. Cross compilers
4. Optimizing compilers

(What are the different types of compilers? | Reference.com, n.d.) A single pass compiler reads each compilation passes through the parts of each compilation unit – combined total package of source code files only once, immediately translating each part into its final machine code, as the name suggests compiler traverses the program only once. Whereas multi-pass compiler converts the program into one or more intermediate representations in steps between source code and machine code, and which reprocesses the entire compilation unit in each sequential pass. Multipass compilers transverses and processes the source code or abstract syntax tree of a program several times. Each pass takes the result of the previous pass as the input, and creates an intermediate output. While most compilers are specific for computer architecture and operating system Cross compilers overcome this and can compile code for other platforms also. Optimizing compilers are used to minimize or maximize some attributes of an executable program, these are used to mostly minimize the time taken to execute a program on mobile devices.

References

(n.d.). Retrieved from wikipedia.com: https://en.wikipedia.org/wiki/Compiler

http://www.tutorialspoint.com. (n.d.). Retrieved from TutorialsPoint: http://www.tutorialspoint.com/computer_programming/computer_programming_environment.htm

Kumar, L. (n.d.). *Compiler Vs Interpreter*. Retrieved from techwelkin.com: http://techwelkin.com/compiler-vs-interpreter

Robert Sedgewick, K. W. (2008). *Introduction to Programming in Java: An Interdisciplinary Approach* (1st ed.). Princeton, New Jersey, USA: Princeton University.

What are the different types of compilers? | Reference.com. (n.d.). Retrieved from https://www.reference.com/technology/different-types-compilers-944f1bc982c6c535