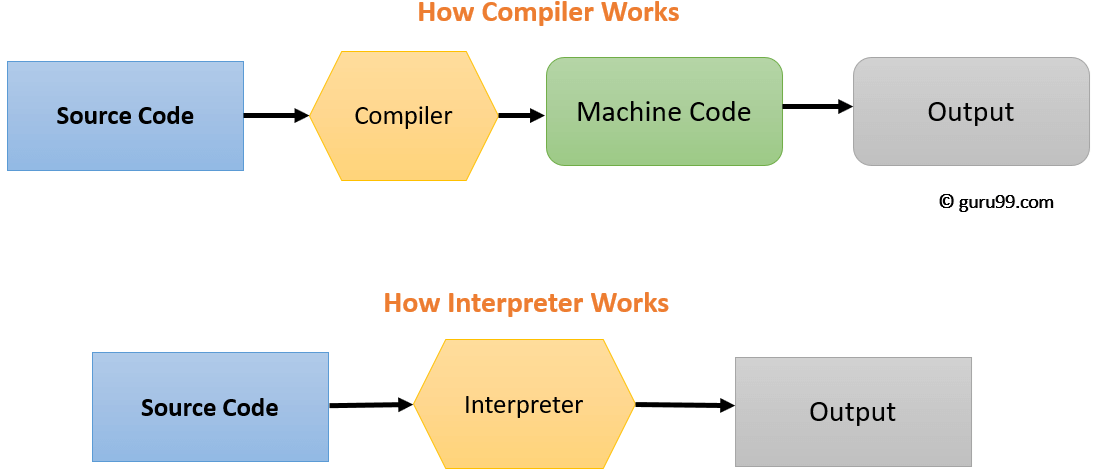
INTERPRETER & COMPLILER MODELS

Programmers write code with the help of high-level languages. But, what does it mean, high-level languages?  A high-level language is that language which understands by people easily. To expand it, I can say that it consists of phrases and words languages in common use usually in English. This is also called source code or source language. But when it comes to computers, they will never understand what does this code mean. Why? Because they have their own language, which is consist of 1 and 0 and called machine or low-level language. That is why we need a translator which will translate our code to the computer in the order it can understand what it really must do. Here interpreter and compiler models come to help us. As you already understood interpreter and compiler languages are translators among the human-readable code and computer. If you read the article until there the questions like:  “But why there is not just one translator, but two?” or “What is the difference between them?” might become to your mind. Answer for first question is they are doing the same work but in different ways and programmer selects one of them depended from which program he or she will write. For answering second question, let’s look at these models separately and differentiate them step-by-step.



INTERPRETER

Interpreter language reads your code and translates one statement at a time, means translates the code while the program running. It stays with you all the time and translate your code immediately, that is why the interpreter exists in the memory during interpretation. It doesn’t generate output program and intermediate machine code, so of course doesn’t save any machine code , because there isn’t any machine code for storing it. It is slow process, because the mechanic each time wait for new instructions, however it also give programmer a chance to correct his or her errors immediately without returning the source code. Remember: Program Execution is a part of Interpretation process, so it is performed line by line. As mentioned before interpreters see code line by line, and thus optimizations are not as robust as compilers, but one of the best advantages of this method that interpreted languages support dynamic typing. It is based on Interpretation Method and this method is more suitable for the program and development environments like web environments, where load times are important. Due to all the exhaustive analysis is done, compiles take relatively larger time to compile even small code that may not be run multiple times. In such cases, interpreters are better.

COMPILER

Compiler language reads the source code all in ones and translate its to the machine code before the program run. Compiler firstly check correctness of the code, if it is not correct gives an error messages and you should correct that error or errors, because the program will never run without fixing errors. If nothing wrong with your code compiler generates output program in the form of exe which is independent from original program and saves machine language as machine code on a disk. After all, this analyses process done, it runs the program. Remember: Program execution is separate from the compilation. It performed only after the entire output program is compiled, that is why \*target program do not require the compiler in the memory. Of course, to analyze the code take some time, but after that process goes very fast no waiting time needed as in interpreter. However, if you make some mistakes on your code it is too late now and for changing your mistake you must go to the source code. One of the advantages of this language is also code optimization. The compiler sees the entire code upfront and performs lots of optimizations that make code run faster. Despite this, unfortunately it does not support dynamic typing as compilers cannot predict what happens at turn time. It is based on language translationlinking-loading model and best suited for the Production Environment. C and C++ are a most popular a programming language which uses compilation model.

\*target program- a fully compiled or assembled program ready to be loaded into the computer.

CONCLUSION

If you still confuse the differences between them or it is hard to you remember which property belong to which language you can simply look at the words themselves. Inter- means between. The interpreter stays between your program and computer during the whole process and translates each statement line-by-line. On the other hand compile means to pile together. Compiler piles together your program and computer, then translates the whole code all in ones and then left you alone. For summarize the information which you read above, you can also use this table as a shortcut.

|  |  |  |
| --- | --- | --- |
| Basis of Comparison | Compiler | Interpreter |
| Input | *whole code* | *single line of code* |
| Output | *intermediate object code* | *no output* |
| Working speed | *Fast* | *Slower* |
| Working mechanism | *The compilation is done before execution.* | *Compilation and execution take place simultaneously.* |
| Analyze speed | *Slower* | *Fast* |
| Memory | *do not require the compiler in the memory* | *require the interpreter in the memory* |
| Displaying errors | *all errors after compilation* | *each error line one by one* |
| Solving Errors | *difficult* | *Easier* |
| Optimization | *yes* | *no* |
| Dynamic typing | *no* | *Yes* |
| Source code is… | *private* | *Public* |
| Pertaining Programming languages | *C, C++, Objective-C* | *PHP, JavaScript* |

INTERMEDIATE APPROACH

As you see both of these methods have their own pros and cons. That is why there is also third way of doing this which is a bit both of them and called Intermediate approach. Instead of compile model where all the work done upfront but can be inflexible or the interpreted model where all the work is done on the receiving end but can a little bit slower, in this method we kind of doing half and half. This is sometimes referred as just-in-time or JIT compilation and sometimes also goes by the name bytecode. So, this process has to happen somehow. It’s just how much of it happens on your computer and much of it happens on my computer.

REFFERENCES

<https://techdifferences.com/difference-between-compiler-and-interpreter.html#KeyDifferences>

<https://www.programiz.com/article/difference-compiler-interpreter>

<https://www.guru99.com/difference-compiler-vs-interpreter.html#:~:text=Compiler%20transforms%20code%20written%20in,while%20interpreted%20code%20runs%20slower>.

<https://www.businessinsider.in/difference-between-compiler-and-interpreter/articleshow/69523408.cms>

<https://www.youtube.com/watch?v=e4ax90XmUBc>

<https://www.youtube.com/watch?v=I1f45REi3k4>