Compiled vs Interpreted

Huy Quang Lai
Department of Computer Science - Texas A&M

created for
CSCE 314: Programming Languages
Prof. Lupoli
12/11/2022

When determining a programming language to use for a project, a question that might arise is whether to use a compiler language or an interpreted language. Before a compiled language can be run, the compiler must compile the entire program beforehand. This contrasts with an interpreted language where the interpreter needs to interpret each line of code one by one as the program executes. This difference in execution leads to the main difference between these two approaches.

Since the compiled languages are compiled before a program is run, the compiler can check for errors in the program such as syntax errors, arithmetic errors, or unreachable code. However, compilers cannot detect runtime errors until the program is run. In contrast, interpreted languages can only detect errors in runtime. In the worst case, a syntax error at the end of the program won't get caught until almost the entire program has executed. Below is sample python code to demonstrate this.

```
x = 0
x += 1
x += 1
x += 1
x += 1
x += 1
x += 1
x += 1
x += 1
x + y
```

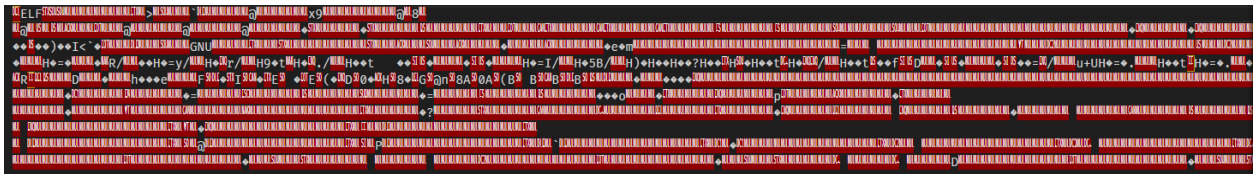
Even though the variable y is undefined, python's interpreter will not catch this error until the interpreter reaches that line. In contrast, compiled languages such as c this error is caught before the program can run.

Another consequence of compiled languages is that if any change is made to the program, the compiler must recompile the entire program. However once compiled, the program can be rerun several times with similar execution time. In contract, interpreted languages have no such luxury and must reinterpret each line of code every time the program is run. This results in

interpreted languages generally being slower than compiled languages as compiled languages do not need to be recompiled before each run.

Because compiled languages need to be compiled before the program is run, each machine needs its own compiler to be able to run the program. As such compiled languages are generally restricted to a select number of machines as a new compiler must be written for each new machine. As an example, c has several different compilers, a full list can be found here. Because of the wide variety of machines, popular languages such as c can grow to have numerous compilers. Interpreted languages, however, are interpreted on the fly and are generally more platform independent as a result. Interpreters only need to be updated when the programming language itself introduces or deprecates techniques and feathers.

Both the compiler and interpreter do the same job, convert the program written into a set of instructions that the machine can understand. Most modern programming languages combine natural languages and mathematical syntax to create high-level programming languages. A machine, however, only operates with binary values and cannot understand what programmers write in the program files. As a result, a translation between the human readable programming language and machine readable languages is needed. Below is an example of raw machine instructions.



Although completely unreadable by humans, a machine will have no trouble understanding what this program does. When run in a UNIX environment, with the command `./a.out`, displays the following to the terminal.

As evidenced by the compiled binary code, even a simple program is completely impossible to read by humans. The code that resulted in this unreadable mess was compiled with gcc and is as follows.

```c
//include standard input and output
#include <stdio.h>
//start the main() program
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    //return 0 to end the program
    return 0;
}
```

Because of the differences between compiled languages and interpreted languages, programmers can select which type of language they would like to develop their projects in. Each type of language has its own benefits and drawbacks.

Interpreted languages allow for dynamic typing unlike their compiled counterparts because of the runtime interpretation of code. With dynamic typing, the data type of variables does not have to be known ahead of time and can allow for more flexibility when developing code. However, with such unlimited flexibility could also cause variables to convert to a type unnecessarily and cause problems later in the program.

Some more advantages of interpreted languages is they can make markup languages and scripting languages much easier than compiled languages. Webpages utilize HTML, CSS, and JavaScript to dynamically alter information displayed to users such as color contracts for the visually impared. Additionally these languages can also perform translations of these same web pages into other languages. Below, is an example of html code that will display “Hello, world” using built in JavaScript.

```
<!DOCTYPE HTML>
<html>
<body>
<p>Before the script...</p>
     <script>
        alert( 'Hello, world!' );
    </script>
    <p>...After the script.</p>
</body>
</html>
```

Some languages blur the line between interpreted and compiled languages and take advantage of features from both aspects. Languages such as python and java are examples of intermediary languages. Python focuses more on its interpreted side, while java focuses more on its compiled side. Some of python's libraries are written in c, taking advantage of python's compiled aspects and performing some level of compile-time execution. Python's interpreted side comes in the form of the Python Virtual Machine that can convert from Python's compiled byte code into machine instructions. Java also has its own Virtual Machine that allows for Java's platform independence.

Each type of language has its own advantages and disadvantages. Depending on the type of program being constructed and the functionality of the program, compiled languages could be more efficient than interpreted languages. In other situations, interpreted languages are more beneficial. These categories are equivalently powerful and one type will not always be better than another.

Sources

https://www.baeldung.com/cs/compiled-vs-interpreted-languages

https://www.freecodecamp.org/news/compiled-versus-interpreted-languages

https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/

https://en.wikipedia.org/wiki/List_of_compilers