Compilation of D Code

This lesson teaches us the process of code compilation and the job of a compiler.

WE'LL COVER THE FOLLOWING

Compilers for D language

As of now, there are *three* D compilers to choose from:

- **dmd:** the Digital Mars compiler
- **gdc:** the D compiler of GCC
- **ldc:** the D compiler that targets the LLVM compiler infrastructure.

In order to avoid the trouble of installing a compiler, we have already set up an environment for you on our platform so that you can easily learn and test your D programming language codes.

Compilation

We have seen that the two tools most used in D programming are the text

editor and the compiler. D programs are written in a text editor like the one we used in the previous lesson to write the code.

Although you don't have to worry about the installation process, it is still a good idea to understand how the compilation works.

Machine code

The brain of the computer is the *microprocessor* (or the *CPU, short for central processing unit*). e

Most CPU architectures use machine code specific to that particular architecture. These machine code instructions are determined under hardware constraints during the design stage of the architecture. At the lowest level, these machine code instructions are implemented as electrical signals. Because the ease of coding is not a primary consideration at this level, writing programs directly in the form of the machine code of the CPU is a very difficult task. These machine code instructions are special numbers, which represent various operations supported by the CPU.

Example

For an imaginary 8-bit CPU, the numbers 4, 5, and 6 might be used to represent various operations as follows:

4 – Loading
5 – Storing
6 – Incrementing

Assuming that the leftmost 3 bits are the operation number and the rightmost 5 bits are the value that is used in that operation, a sample program in machine code for this CPU might look like the following:

Operation	Value	Meaning
100	11110	LOAD 11110
101	10100	STORE 10100
110	10100	INCREMENT 10100
000	00000	PAUSE

Machine code for the example 8 bit CPU

Being so close to the hardware, machine code is not suitable for representing higher-level concepts like a playing card or a student record.

Programming language

Programming language

Programming languages are designed as efficient ways of programming a CPU, capable of representing higher-level concepts. Programming languages do not have to deal with hardware constraints; their main purposes are ease of use and expressiveness. Programming languages are easier for humans to understand and closer to natural languages.
For example:

```
if (a_card_has_been_played()) {
   display_the_card();
}
```

A simple programming language code

However, programming languages adhere to much more strict and formal rules than any spoken language. Moreover, high-level language instructions need to be translated to machine code for execution by the CPU.

Interpreter

An **interpreter** is a tool (a program) that reads source code instructions from source code and executes them. For example, in the code above, an interpreter would understand to first execute `a_card_has_been_played()` and then conditionally execute `display_the_card()`. From the point of view of the programmer, executing with an interpreter involves just two steps:

- writing the source code
- giving the source code to the interpreter

The interpreter must read and understand the instructions every time the program is executed. For this reason, running a program with an interpreter is usually slower than running the compiled version of the same program. Additionally, interpreters usually perform very little analysis on the code before executing it. As a result, most interpreters discover programming errors only after they start executing the program.

Compiler

A **compiler** is another tool that reads the instructions of a program from source code. From the point of view of the programmer, executing code with a

compiler involves three steps:

- writing the source code
- compiling the source code
- running the produced program

How is a compiler different from an interpreter?

- The compiler does not execute the code; rather, it produces a program written in another language (usually machine code). This produced program is responsible for the execution of the instructions that were written by the programmer.
- Unlike an interpreter, the compiler reads and understands the source code only once, during compilation. For that reason and in general, a *compiled program runs faster as compared to executing that program with an interpreter*.
- Compilers usually perform advanced analysis on the code, which helps with producing fast programs and catching programming errors before the program even starts running.

Disadvantages of using a compiler

- Having to compile a program every time it is changed is complicated and a potential source of human errors like running the wrong compilation command.
- The programs that are produced by a compiler can usually run only on a specific platform. To run a program on a different kind of processor or on a different operating system, the program would have to be recompiled.
- Additionally, the languages that are easy to compile are usually less dynamic than those that run on an interpreter.

For reasons like safety and performance, some languages have been designed to be compiled. Ada, C, C++, and D are some of them.

Compilation error

As the compiler compiles a program according to the rules of the language, it stops the compilation as soon as it comes across illegal instructions. **Illegal instructions** are the ones that are outside the specifications of the language.

Problems like a mismatched parenthesis, a missing semicolon, a misspelled

keyword, etc. all cause compilation errors. The compiler may also issue a compilation warning when it sees a suspicious piece of code that is not necessarily an error but may cause concern. However, warnings almost always indicate an actual error or bad style, so it is a common practice to consider most or all warnings as errors.

In the next lesson, you will take a quiz to test what you have learned so far.