The 1950s and 60s saw the introduction of high-level languages, such as Fortran and Algol.
These languages provide mechanisms, such as subroutines and conditional looping constructs, which greatly enhance the structure of a program, making it easier to express the progression of instruction execution; that is, easier to visualise program flow. Also, these mechanisms are an abstraction of the underlying machine instructions and, unlike assembler, are not tied to any particular hardware.
Thus, ideally, a program written in a high-level language may be ported to a different machine and run without change. To produce executable code from such a program, it is translated to machinespecific assembler language by a *compiler* program, which is then coverted to machine code by an assembler (see Appendix B for details on the compilation process).
Compiled code is not the only way to execute a high-level program. An alternative is to translate the program on-the-fly using an *interpreter* program (e.g., Matlab, Python, etc). Given a text-file containing a high-level program, the interpreter reads a high-level instruction and then executes the necessary set of low-level operations. While usually slower than a compiled program, interpreted code avoids the overhead of compilation-time and so is good for rapid implementation and testing.
Another alternative, intermediate between compiled and interpreted code, is provided by a *virtual machine* (e.g., the Java virtual machine) which behaves as an abstract-machine layer on top of a real machine. A high-level program is compiled to a special *byte-code* rather than machine language, and this intermediate code is then interpreted by the virtual machine program. Interpreting byte code is usually much faster than interpreting high-level code directly. Each of these representations has is relative advantages: compiled code is typically fastest, interpreted code is highly portable and quick to implement and test, and a virtual machine offers a combination of speed and portability.
The primary purpose of a high-level language is to permit more direct expression of a programmer's design. The algorithmic structure of a program is more apparent, as is the flow of information between different program components. High-level code modules can be designed to "plug" together piece-by-piece, allowing large programs to be built out of small, comprehensible parts. It is important to realise that programming in a high-level language is about communicating a software design to programmers *not* to the computer. Thus, a programmer's focus should be on modularity and readability rather than speed. Making the program run fast is (mostly) the compiler's concern.

Development of "C" (history)

The C programming language is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at AT & T Bell Laboratories of USA. C was originally first implemented on the DEC PDP-11 computer in 1972.
In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.