Programmable devices have existed for centuries. These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. Programs were mostly entered using punched cards or paper tape. However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory. Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. New languages are generally designed around the syntax of a prior language with new functionality added, (for example C++ adds object-orientation to C, and Java adds memory management and bytecode to C++, but as a result, loses efficiency and the ability for low-level manipulation). These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. One approach popular for requirements analysis is Use Case analysis. Unreadable code often leads to bugs, inefficiencies, and duplicated code. Ideally, the programming language best suited for the task at hand will be selected. However, readability is more than just programming style. Use of a static code analysis tool can help detect some possible problems. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. FORTRAN, the first widely used high-level language to have a functional implementation, came out in 1957, and many other languages were soon developed—in particular, COBOL aimed at commercial data processing, and Lisp for computer research. Auxiliary tasks accompanying and related to programming include analyzing requirements, testing, debugging (investigating and fixing problems), implementation of build systems, and management of derived artifacts, such as programs' machine code. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Code-breaking algorithms have also existed for centuries. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages.