Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.  
In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages.  
There exist a lot of different approaches for each of those tasks.  
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.  
Use of a static code analysis tool can help detect some possible problems.  
Many applications use a mix of several languages in their construction and use.  
Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages.  
Compiling takes the source code from a low-level programming language and converts it into machine code.  
As early as the 9th century, a programmable music sequencer was invented by the Persian Banu Musa brothers, who described an automated mechanical flute player in the Book of Ingenious Devices.  
To produce machine code, the source code must either be compiled or transpiled.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
Use of a static code analysis tool can help detect some possible problems.  
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.