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)..  
Programmers typically use high-level programming languages that are more easily intelligible to humans than machine code, which is directly executed by the central processing unit.  
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.  
Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists.  
For example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash.  
 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.  
There exist a lot of different approaches for each of those tasks.  
This can be a non-trivial task, for example as with parallel processes or some unusual software bugs.  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
 Allen Downey, in his book How To Think Like A Computer Scientist, writes:  
 Many computer languages provide a mechanism to call functions provided by shared libraries.  
 The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging).  
 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.  
 Some languages are very popular for particular kinds of applications, while some languages are regularly used to write many different kinds of applications.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
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.