Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
There exist a lot of different approaches for each of those tasks.  
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.  
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.  
Proficient programming thus usually requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.  
It is usually easier to code in "high-level" languages than in "low-level" ones.  
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.  
Many applications use a mix of several languages in their construction and use.  
Normally the first step in debugging is to attempt to reproduce the problem.  
Use of a static code analysis tool can help detect some possible problems.  
Ideally, the programming language best suited for the task at hand will be selected.  
 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.  
To produce machine code, the source code must either be compiled or transpiled.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
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.