In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers. It is very difficult to determine what are the most popular modern programming languages. Use of a static code analysis tool can help detect some possible problems. Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. 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. Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug. For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. The Unified Modeling Language (UML) is a notation used for both the OOAD and MDA. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. Scripting and breakpointing is also part of this process. Code-breaking algorithms have also existed for centuries. Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. It is usually easier to code in "high-level" languages than in "low-level" ones. 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. 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). Ideally, the programming language best suited for the task at hand will be selected. Integrated development environments (IDEs) aim to integrate all such help. After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.