High-Level languages

- A high-level programming language is a programming language with strong abstraction from the details of the computer.
- It may use natural language elements, be easier to use, or be from the specification of the program, making the process of developing a program simpler and more understandable with respect to a low-level language.
- The amount of abstraction provided defines how "high-level" a programming language is.
- Translated into machine code before being run.

Compilers

- A program that translates a high-level language program into machine code.
- Originally output of a compiler was the assembly language version of the program which was then converted into machine code using assembler.
- Modern compilers can directly output machine code.
- A compiler itself is a program, so a machine code version of the compiler must be available.

Compilers

- If the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs, the compiler is known as a cross-compiler.
- A program that translates from a low level language to a higher level one is a decompiler.
- A program that translates between high-level languages is usually called a language translator, source to source translator, or language converter.

Compilers

- A language rewriter is usually a program that translates the form of expressions without a change of language.
- A compiler is likely to perform many or all of the following operations:
  - Lexical analysis
  - Preprocessing
  - Parsing
  - Code generation
  - Code optimization.

Compilers

- Program faults caused by incorrect compiler behavior can be very difficult to track down and work around; therefore, compiler implementors invest a lot of time ensuring the correctness of their software.
- The term compiler-compiler is sometimes used to refer to a parser generator, a tool often used to help create the lexer and parser.

Compilers

- Compilers bridge source programs in high-level languages with the underlying hardware.
- A compiler requires
  - determining the correctness of the syntax of programs
  - generating correct and efficient object code
  - run-time organization
  - formatting output according to assembler and/or linker conventions.
- A compiler consists of three main parts: the frontend, the middle-end, and the backend.

Compilers

- The front end checks whether the program is correctly written in terms of the programming language syntax and semantics.
  - Here legal and illegal programs are recognized.
  - Errors are reported, if any, in a useful way.
  - Type checking is also performed by collecting type information.
  - The frontend then generates an intermediate representation or IR of the source code for processing by the middle-end.

Compilers

- The middle end is where optimization takes place.
  - Typical transformations for optimization are removal of useless or unreachable code, discovery and propagation of constant values, relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computation based on the context.
  - The middle-end generates another IR for the following backend.
  - Most optimization efforts are focused on this part.

Compilers

- The back end is responsible for translating the IR from the middle-end into assembly code.
  - The target instruction(s) are chosen for each IR instruction.
  - Register allocation assigns processor registers for the program variables where possible.
  - The backend utilizes the hardware by figuring out how to keep parallel execution units busy, filling delay slots, and so on.

Interpreters

- A program that inputs a program in a high-level language and directs the computer to perform the actions in each statement.
  - Programs are 'indirectly' executed ("interpreted") by an interpreter program.
- Initially, interpreted languages were compiled line-by-line.
  - Each line was compiled as it was about to be executed, and if a loop or subroutine caused certain lines to be executed multiple times, they would be recompiled every time.

Interpreters

- Modern interpreted languages use an intermediate representation, which combines compiling and interpreting.
- In this case, a compiler may output some form of bytecode or threaded code, which is then executed by a bytecode interpreter.
- The intermediate representation can be compiled once and for all (as in Java), each time before execution (as in Perl or Ruby), or each time a change in the source is detected before execution (as in Python).

Interpreters

- Features that are often easier to implement in interpreters than in compilers include (but are not limited to):
  - platform independence
  - reflection
  - dynamic typing
  - smaller executable program size
- The main disadvantage of interpreting is a much slower speed of program execution compared to direct machine code execution on the host CPU.

Interpreters

- Bytecode, also known as p-code (portable code), is a term which has been used to denote various forms of instruction sets designed for efficient execution by a software interpreter as well as being suitable for further compilation into machine code.
- Bytecode may often be either directly executed on a virtual machine (i.e. interpreter), or it may be further compiled into machine code for better performance.

Interpreters

- Unlike human-readable source code, bytecodes are compact numeric codes, constants, and references (normally numeric addresses) which encode the result of parsing and semantic analysis of things like type, scope, and nesting depths of program objects.
- They therefore allow much better performance than direct interpretation of source code.

Interpreters

- A bytecode program may be executed by parsing and directly executing the instructions, one at a time.
  - This kind of bytecode interpreter is very portable.
- Dynamic translators, or "just-in-time" (JIT) compilers, translate bytecode into machine language as necessary at runtime.
  - This makes the virtual machine unportable, but doesn't lose the portability of the bytecode itself.

Interpreters

- Just-in-time compilation (JIT), also known as dynamic translation, is a method to improve the runtime performance of computer programs.
- JIT compilers represent a hybrid approach, with translation occurring continuously, as with interpreters, but with caching of translated code to minimize performance degradation.
- Offers other advantages over statically compiled code at development time, such as ability to enforce security guarantees.
- JIT typically causes a slight delay in initial execution of an application, due to the time taken to load and compile the bytecode.

Interpreters

- A common goal of using JIT techniques is to reach or surpass the performance of static compilation, while maintaining the advantages of bytecode interpretation.
  - Parsing the original source code and performing basic optimization is often handled at compile time, prior to deployment
    - Compilation from bytecode to machine code is much faster than compiling from source.
  - Since the runtime has control over the compilation, like interpreted bytecode, it can run in a secure sandbox.

Interpreters

- JIT code generally offers far better performance than interpreters.
- In addition, it can in some cases offer better performance than static compilation, as many optimizations are only feasible at run-time:
  - The compilation can be optimized to the targeted CPU and the operating system model where the application runs.
  - The system is able to collect statistics about how the program is actually running in the environment it is in, and it can rearrange and recompile for optimum performance.

4GL

- A fourth-generation programming language (1970s-1990) (abbreviated 4GL) is a programming language or programming environment designed with a specific purpose in mind, such as the development of commercial business software.
- In the history of computer science, the 4GL followed the 3GL in an upward trend toward higher abstraction and statement power.
- 4GL projects are more oriented toward problem solving and systems engineering.