Homework 1

Braulio Millan Chin
Programming Rob2A

Stages of program compilation

While there are many different programming languages with many different compilators to go with them, the steps can be summarized as follows:

Lexical analysis

The first part of the lexical analysis operation is removal of any non-program elements, like indents on white space and comments on the code. These are helpful to the programmer but are not necessary for the executable code, so the compiler removes them during lexical analysis.

Syntax analysis

Once tokens have been assigned to the code elements, the compiler checks that the tokens are in the correct order and follow the rules of the language. During this stage, an abstract syntax tree (AST) is created. This maps the structure of the program, first dropping the brackets, semicolons, etc., that were used by the programmer. If required tokens are missing from the tree, or in the wrong place, the compiler will report an error.

Code generation

Once the semantic analysis phase is over, the compiler generates intermediate (assembly) code. This intermediate code makes it easy to translate the source code into binary (object) code. After the compiler creates one or more object files, then another program called the linker kicks in. The job of the linker is as follows:

- To take all the object files generated by the compiler and combine them into a single executable program.
- The linker also links library files used and required by the source code.
- The linker makes sure all cross-file dependencies are resolved properly.

For example, if you define something in one .cpp file, and then use it in another .cpp file, the linker connects the two together. If the linker is unable to connect a reference to something, you'll get a linker error, and the linking process will abort.

Code optimization

The optimizer may identify redundant or repeated code, and remove or rearrange the code as necessary. The main goal of this phase is to improve on the intermediate code to generate a code that runs faster and occupies less space.

Levels of programming

Abstraction is a simplified version of something technical, such as a function or an object in a program. The goal of "abstracting" data is to reduce complexity by removing unnecessary information. Some programming languages provide less or no abstraction while some provide higher abstraction. They can be classified based on the levels of abstraction.

Low-level language

Low-level language is a programming language that provides no abstraction from the hardware.

Machine code

Machine language, or machine code, is a low-level language comprised of binary digits (ones and zeros). Since computers are digital devices, they only recognize binary data. Every program, video, image, and character of text is represented in binary. This binary data, or machine code, is processed as input by the CPU. The resulting output is sent to the operating system or an application, which displays the data visually.

While machine code is comprised of 1s and 0s, different processor architectures use different machine code. A compiler must compile high-level source code for the correct processor architecture in order for a program to run correctly.

Assembly code

An assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware. Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.

Low-level programming languages such as assembly language are a necessary bridge between the underlying hardware of a computer and the higher-level programming languages

Intermediate level language

Intermediate language is an abstract programming language used by a compiler as an in-between step when translating a computer program into machine code.

High level language

High-level languages allow programmers to write instructions in a language that is easier to understand than low-level languages. Translators are needed to translate programs written in high-level languages into the machine code that a computer understands. They allow the programmer to focus on what needs to be done, rather than on how the computer actually works.

An example of high-level languages are most modern languages being used today by many programmers, including but not limited to: Python, C / C# / C++, Java, JavaScript, etc.

Bibliography

Stages of program compilation

Stages of compilation. Isaac Computer Science. Retrieved 9 May 2021, from https://isaaccomputerscience.org/concepts/sys_trans_stages.

Phases of Compiler with Example. Guru99.com. (2021). Retrieved 9 May 2021, from https://www.guru99.com/compiler-design-phases-of-compiler.html.

Doe, A. (2018). Introduction to the compiler, linker, and libraries. Learncpp.com. Retrieved 9 May 2021, from https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/.

Levels of programming

Abstraction. Techterms.com. (2019). Retrieved 9 May 2021, from https://techterms.com/definition/abstraction.

Machine Language. Techterms.com. (2019). Retrieved 9 May 2021, from https://techterms.com/definition/machine_language.

Hope, C. (2018). What is an Intermediate Language. Computerhope.com. Retrieved 9 May 2021, from https://www.computerhope.com/jargon/i/il.htm.

Fernando, J. (2020). Assembly Language. Investopedia. Retrieved 9 May 2021, from https://www.investopedia.com/terms/a/assembly-language.asp.

Types of programming language. BBC Bitesize. Retrieved 9 May 2021, from https://www.bbc.co.uk/bitesize/guides/z4cck2p/revision/1.