CCA-101: Fundamentals of IT & Programming

Assignment – 2

Q.1 What is the difference between Machine Language and High Level Language?

Ans. **Machine language**

Updated: 06/30/2019 by Computer Hope

Sometimes referred to as machine code or object code, machine language is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding.

The exact machine language for a program or action can differ by operating system. The specific operating system dictates how a compiler writes a program or action into machine language.

Computer programs are written in one or more programming languages, like C++, Java, or Visual Basic. A computer cannot directly understand the programming languages used to create computer programs, so the program code must be compiled. Once a program's code is compiled, the computer can understand it because the program's code is turned into machine language.

Machine language example

Below is an example of machine language (binary) for the text "Hello World."

```
01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100
```

Below is another example of machine language (non-binary), which prints the letter "A" 1000 times to the computer screen.

```
169 1 160 0 153 0 128 153 0 129 153 130 153 0 131 200 208 241 96
```

High-level programming language

From Wikipedia, the free encyclopedia

Jump to navigationJump to search

In computer science, a high-level programming language is a programming language with strong abstraction from the details of the computer. In contrast to low-level programming languages, it may use natural language *elements*, be easier to use, or may automate (or even hide entirely) significant areas of computing systems (e.g. memory management), making the process of developing a program simpler and more understandable than when using a lower-level language. The amount of abstraction provided defines how "high-level" a programming language is.[1]

In the 1960s, high-level programming languages using a compiler were commonly called autocodes.[2] Examples of autocodes are COBOL and Fortran.[3]

The first high-level programming language designed for computers was Plankalkül, created by Konrad Zuse.[4] However, it was not implemented in his time, and his original contributions were largely isolated from other developments due to World War II, aside from the language's influence on the "Superplan" language by Heinz Rutishauser and also to some degree Algol. The first significantly widespread high-level language was Fortran, a machine-independent development of IBM's earlier Autocode systems. Algol, defined in 1958 and 1960 by committees of European and American computer scientists, introduced recursion as well as nested functionsunder lexical scope. It was also the first language with a clear distinction between value and name-parameters and their corresponding semantics.[5] Algol also introduced several structured programming concepts, such as the while-do and if-then-else constructs and its syntax was the first to be described in formal notation – "Backus–Naur form" (BNF). During roughly the same period, Cobol introduced records (also called structs) and Lisp introduced a fully general lambda abstraction in a programming language for the first time.

Contents

Features[edit]

"High-level language" refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses, and call stacks, high-level languages deal with variables, arrays, objects, complex arithmetic or boolean expressions, subroutines and functions, loops, threads, locks, and other abstract computer science concepts, with a focus on usability over optimal program efficiency. Unlike low-level assembly languages, high-level languages have few, if any, language elements that translate directly into a machine's native opcodes. Other features, such as string handling routines, object-oriented language features, and file input/output, may also be present. One thing to note about high-level programming languages is that these languages allow the programmer to be detached and separated from the machine. That is, unlike

low-level languages like assembly or machine language, high-level programming can amplify the programmer's instructions and trigger a lot of data movements in the background without their knowledge. The responsibility and power of executing instructions have been handed over to the machine from the programmer.

Abstraction penalty[edit]

High-level languages intend to provide features which standardize common tasks, permit rich debugging, and maintain architectural agnosticism; while low-level languages often produce more efficient code through optimization for a specific system architecture. *Abstraction penalty* is the cost that high-level programming techniques pay for being unable to optimize performance or use certain hardware because they don't take advantage of certain low-level architectural resources. High-level programming exhibits features like more generic data structures and operations, run-time interpretation, and intermediate code files; which often result in execution of far more operations than necessary, higher memory consumption, and larger binary program size.[6][7][8] For this reason, code which needs to run particularly quickly and efficiently may require the use of a lower-level language, even if a higher-level language would make the coding easier. In many cases, critical portions of a program mostly in a high-level language can be hand-coded in assembly language, leading to a much faster, more efficient, or simply reliably functioning optimised program.

However, with the growing complexity of modern microprocessor architectures, well-designed compilers for high-level languages frequently produce code comparable in efficiency to what most low-level programmers can produce by hand, and the higher abstraction may allow for more powerful techniques providing better overall results than their low-level counterparts in particular settings.[9] High-level languages are designed independent of a specific computing system architecture. This facilitates executing a program written in such a language on any computing system with compatible support for the Interpreted or JIT program. High-level languages can be improved as their designers develop improvements. In other cases, new high-level languages evolve from one or more others with the goal of aggregating the most popular constructs with new or improved features. An example of this is Scala which maintains backward compatibility with Java which means that programs and libraries written in Java will continue to be usable even if a programming shop switches to Scala; this makes the transition easier and the lifespan of such high-level coding indefinite. In contrast, low-level programs rarely survive beyond the system architecture which they were written for without major revision. This is the engineering 'trade-off' for the 'Abstraction Penalty'.

Q.2 Discuss about different data types of C programming Language.

Ans.

Data Type

A data type is a type of data. Of course, that is rather circular definition, and also not very helpful. Therefore, a better definition of a data type is a data storage format that can contain a specific type or range of values.
When computer programs store data in variables, each variable must be assigned a specific data type. Some common data types include integers, floating point numbers, characters, strings, and arrays. They may also be more specific types, such as dates, timestamps, boolean values, and varchar (variable character) formats.

Some programming languages require the programmer to define the data type of a variable before assigning it a value. Other languages can automatically assign a variable's data type when the initial data is entered into the variable. For example, if the variable "var1" is created with the value "1.25," the variable would be created as a floating point data type. If the variable is set to "Hello world!," the variable would be assigned a string data type. Most programming languages allow each variable to store a single data type. Therefore, if the variable's data type has already been set to an integer, assigning string data to the variable may cause the data to be converted to an integer format.

Data types are also used by database applications. The fields within a database often require a specific type of data to be input. For example, a company's record for an employee may use a string data type for the employee's first and last name. The employee's date of hire would be stored in a date format, while his or her salary may be stored as an integer. By keeping the data types uniform across multiple records, database applications can easily search, sort, and compare fields in different records.

Q3. Find the output of the following expressions

a) X=20/5*2+30-5 b) Y=30 – (40/10+6) +10 c) Z= 40*2/10-2+10

Ans.

a) X=20/5*2+30-5

Q4. Describe the syntax of the following statements

a) If – else statement b) for loop c) while loop d) do-while loop

Ans.

Q5. Find the output of the following program segments.

Ans.