Competitive Exams: Programming Languages

Programming languages can be divided into three broad categories: Machine languages, assembly languages, and high level languages.

Machine Language

Machine language, also called binary representation, is the only language that the computer can directly execute. Programs written in any other type; of language must be translated into machine language before they can executed. Data represented in binary form is stored in the computer as a series of "of£ and" offf states of electronic devices representing binary digits (or bits). An "ton" bit indicates the presence of an electric current, whereas an "off" bit indicates its absence.

A programmer writing instructions in machine language can specify an "on" bit with the numeral 1 and an" off bit with the numeral 0. Every operation that the computer is capable of performing (such as addition or storing a value in a given memory location) is indicated by a specific binary code. The programmer must use the proper code for each operation. Also, the memory locations must be accessed by listing their storage addresses in binary code in theJnstruction.

Because these series of Is and Os have no intrinsic meaning to humans, the writing of this type of code is very error-prone. The programmer must carefully keep track of which values have been stored in which storage locations. It is easy to refer to an incorrect location or to accidentally store a new value in a location that has already been used for something else, thereby losing the previous contents.

Machine language is different for each kind of computer, so the programmer must be familiar with the particular computer for which he or she is programming. Although machine language programs are very tedious to write, they allow the programmer to fully utilize the computer's potential because he or she is interacting directly with the computer hardware.

When computers were first developed, machine language was the only way they could be programmed. The people who worked with these computers quickly realized met this method often led to programming errors and were extremely time-consuming for the programmer. In a program of any significant length, machine language programming is a difficult, tedious task.

Assembly Languages

Assembly languages, also referred to as low-level languages, were developed to make programming easier. In assembly language, the programmer uses symbolic names (rather than Is and 0s) to specify various machine operations. For example, the word ADD might be used to instruct the computer to add'the contents of two storage locations. The use of these symbolic names makes assembly language programming easier than machine language programming. Another important improvement over machine language. Is the use of names to represent storage locations, so that the programmer no longer has to know the address of the storage location in which a particular value is kept. In general, assembly language instructions have a one-to-one correspondence with machine language instructions.

The assembly language instructions that perform the task of adding two numbers might look like this:

These instructions tell the computer to perform the following steps:

1. Get the value contained in address A and put it in a location in the ALU (called an accumulator).
2. Get the value contained in address B, and add that value to the value in the accumulator.
3. Get the new value in the accumulator and store it in location C.

As with machine languages, assembly languages are different for each kind of computer. Assembly language is easier for people to understand than machine language, but it usually doesn't use computer time as efficiently because assembly instructions mujit be translated into machine language before the computer can execute the instructions. This translaltion is accomplished by a special system program called an assembler. One advantage of assembly language is that because it is specific to the computer1 system being used, the program can be tailored to the system. Often assembly language instructions will execute faster than those written in high level languages.

High-Level Languages

Although it is simpler to write a program in assembly language than in machine language, considerable knowledge of the internal operations of the computer is still required. To simplify programming further, other languages have been developed which resemble “natural language” such as English even more closely. These languages do not require the programmer to understand the technical in details of internal computer operations. Because these languages are strongly oriented toward the programmer rather than toward the computer, they are termed high-level languages, Even a person who does not know a particular high-level language can often determine the general purpose of the program statements. Consider the Pascal statement C = A + B

Even a non-programmable would have little difficulty in understanding that this statement adds the value stored in location A to the value stored in location B

and places the result in location C.

One of the main advantages of high-level languages is that they are usually portable. This means that a program written in one of these languages can usually be run on a variety of computers with minimal changes. But, as with assembly language, high-level language programs must be translated into machine language before the computer can execute them. This task is performed by one of two types of system programs: An interpreter or a compiler.

There are many high-level languages. A few of the more popular ones are COBOL, BASIC, FORTRAN, and Pascal.

Two high-level languages in widespread use in engineering/mathematical/scientific and in business circles respectively, are FORTRAN, an acronym for Formula Translator, and COBOL, an acronym for Common, Business-Oriented Language. The former was developed in the late 1950's by Dr. J. Backus at IBM. The latter was developed by Commander G. M. Hopper of the US, Navy, and is often referred to as COBOL60, because its specifications were first published in 1960. These were among the earliest high level languages; both have undergone substantial revision over the past 20 years or so, and although they do not incorporate many of the data and control-structuring features thought to be desirable in a “good” modern language, they are still among the most commonly used.

Three other high-level languages which are used in the industry to varying extents are RPG (for Report-Program Generator), PL/1 (for Programming Language 7) and BASIC (For Beginners, All-Purpose Symbolic Instruction Code). RPG is related to COBOL, and is meant for similar applications. PL/1 was developed in the late 1960's by IBM as an intended successor to FORTRAN and COBOL-It was referred to as an omnibus language because it was aimed at both scientific and commercial applications. BASIC was developed up to 1967 at Dartmouth College, USA, as a teaching language, and for use in time-sharing, interactive systems.

In 1960, a report on a language (for Algorithmic Language) was published by a committee of European and American computer scientists. This language was the ancestor of-a series of languages known collectively as the Algol-like languages. Two important such languages are Pascal and Algol68. These two languages are in marked contrast. Whereas the former, which was designed primarily for teaching programming, is fairly simple, but still powerful, the latter, is much more complex.

Recently, a competition was held by the US Department of Defense for the design of a new language to be adopted as. a standard in US defense establishments. The competition was won by a team from CII Honey well-Bull in-France. Their language is called-ADA, after the Countess of Lovelace, Lord Byron's daughter, who was a colleague of Charles Babbage and who is credited as being the world's first computer programmer.

Defining a High-Level Language

Since high-level language are much more complex than low-level ones, it is necessary to have powerful, systematic means of describing them. Such a description involves syntax and semantics.

Syntax

The syntax of a language consists of the rules governing the way programs are constructed from the characters'of the language. In the case of Pascal, these characters are letters, digits, space, the operator symbols +, V/, =, <> > and the colon, comma, semicolon, period, left and right parentheses (and), left and right brackets [and], the comment brackets { and }, and the symbol

The syntax of a lantguage can be described using syntax diagrams such as those for Pascal. Syntax diagrams were invented by Professor Wirth to describe the syntax of his programming language.

An older syntactic notation, called Backus Naur Form (BNF) after J. Backus and P. Naur, who developed it for describing the syntax of Algol 60. The description specifies the syntactic classes expression

- term
- factor
- adding operator multiplying operator identifier
- unsigned integer constant digit
- letter or digit