Programming Languages: First Generation

Article provided by: http://www.snappy-apps.com.au/

PROGRAMMING LANGUAGES

A computer program is a set of instructions or codes written driven the functioning of the computer on execution. All computer software's are developed as programs. Programming languages enable users to develop software applications to carry out specific information processing task. Programming languages can be described in terms of their historical position in the development of computer programming systems. Let us study the programming languages according to their generations.

FIRST GENERATION: Early computer systems were programmed using machine language that consisted of strings of binary digits. With this, programmers required advanced technical skill order to develop and enter programs. And as such programs were considered expensive to develop as they took extremely long period to design, cod and test.

Second Generation: Assembly language represented an attempt to simplify the process of creating the computer programs. Symbols and abbreviations were used to create sequences of instructions. An assembly was used to translate a completed assembly program into machine code required by the computer.

Although assembly language provided several advantages over machine language, it also suffered from two major disadvantages. First, since assembly programs did not run as quickly as their machine language counterparts, they are unsuitable for certain tasks, such as those involving large scale data

processing. Second, since programmers were still working at a very low level with the computer's hard ware, it remained difficult to create large or complex programs using assembly language.

THIRD GENERATION: Third generation programming languages provided a more natural means of develop programs by enabling users to create programs made from English-like statements, such programming languages are still in used today and are called high- level languages. Languages such as COBOL allow users to develop programs quickly and easily, although the resulting applications were sometimes slow and efficient.

FOURTH GENERATION: A drive towards even greater ease of use resulted in the development of new programming systems designed to allow even non-technical users to develop their own applications. The focus of such tools was on ease of use and the rapid development of applications. Examples of common programming tools include report generators, query languages and application generators.

FIFTH GENERATION: Fifth-generation languages are likely to base around several different technologies and techniques. As an example, artificial intelligence (AI) method attempt to make a computer system behave the same way as human being. One application for AI is natural language processing, where users can communicate with computer system using English - Like statements. Development in this area may result in programming system that accepts a spoken question from a user and then generate a computer program intended to produce the required the information.

CLASSIFICATION OF PROGRAMMING LANGUAGES

Low-Level Language Vs High- Level Language

A low-level language requires the programmer to work directly with the hardware of the computer system. Instructions are normally entered in machine code or assembly language. Programmers must have detailed knowledge of the computer hardware being used to construct programs. One of the main characteristics of low level languages is that applications run very quickly.

A high-level language allows programmers to issue instructions in a more natural form. Programs are normally created using English - Like statements that can be organized to form a clear logical structure. In addition, high-level languages provide programmers with a variety of tools that assist in the process of creating the program and locating error.

although simpler to use than the machine code or assembly language applications developed, using high-level languages are considered slow. Java, C++ and Visual Basic are high - level languages.

Machine Oriented Vs Problem Oriented: It can be argued that all computer programs are created to serve a specific need. In most cases, programs are constructed to solve a particular problem or meet a clearly defined set of information processing requirements. The way in which programming languages allow problem or set of information processing requirements to be expressed provides a second mean of classification.

A machine - oriented language focuses on the requirements of the computer hardware being used, where programs are produced in a form that suits the way in which the microprocessor functions or operates.

A problem - oriented language focused on the expression of a problem or set of information processing requirements. The language will provide a variety of features that allow programmers to express their

requirements in a natural form. The Language will also provide the facility to translate the program into a form suitable for the computer's microprocessor. Effectively, problem - oriented languages enable programmers to focus on the problem to be solved, rather than on the generation of the computer's hardware.

COMPILER Vs INTERPRETER: The instructions contained within any computer program must be converted into machine language before they can be executed. The way in which instructions are translated into machine language provides a third way classifying programming languages.

COMPILER: The instructions that make up a computer program are often stored as simple text file, usually called a source code file. The instructions held as source code cannot be executed directly by the microprocessor since they must first be converted into machine language. A compiler produces an executable program by converting instructions held as source code into machine language. If the source code is altered in any way, it must be complied again so that a new executable program can be produced.

INTERPRETER: As the program runs, each instruction is taken in turn and converted into machine language by a command interpreter. Since the process of converting each instruction into machine language can take great deals of time, interpreted programs operate much more slowly than compiled programs.