**Generations of Programming Languages**

Computer professionals talk about **levels** or **generations** of programming languages, ranging from “low” to “high.” Programming languages are called **lower level** when they are closer to the language the computer itself uses. The computer understands the **0**s and **1**s that make up bits and bytes. Programming languages are called **higher level** when they are closer to the language humans use – that is, for English speakers, more like English.

There are five generations of programming languages: (1) machine languages, (2) assembly languages, (3) procedural languages, (4) task-oriented languages, and (5) problem and constraint languages.

**Machine Languages: The First Generation**

As we know, a byte is made up of bits, consisting of **1**s and **0**s. These **1**s and **0**s may correspond to electricity’s being **on** or **off** in the computer. They also may correspond to a magnetic charge being present or absent on storage media such as disk or tape. From this two-state system, coding schemes have been developed that allow us to construct letters, numbers, punctuation marks, and other special characters. Examples of these coding schemes, as we saw, are ASCII, EBCDIC, and Unicode.

Data represented in **1**s and **0**s is said to be written in **machine language.** To see how hard this is to understand, imagine if you had to code this:

**111100100111001111010010000100000111000000101011**

Machine languages also vary according to make of computer – another characteristic that makes them hard to work with.

**Assembly Languages: The Second Generation**

Before a computer can process or run any program, the program must be converted or translated into machine language. **Assembly languages** use abbreviations or mnemonics such as ADD that are automatically converted to the appropriate sequence of **1**s and **0**s. Compared to machine languages, assembly languages are much easier for humans to understand and to use. The machine language code we gave above could be expressed in assembly language as **ADD 210(8,13),02B(4,7)**

This is still pretty obscure, of course, and so assembly language is also considered low level. Assembly languages also vary from computer to computer. With the third generation, we advance to high-level languages, many of which are considered **portable languages.** That is, they can be run on more than one kind of computer – they are “portable” from one machine to another.

**High-Level Procedural Languages: The Third Generation**

People are able to understand languages that are more like their own (e.g., English) than machine languages or assembly languages. These more Englishlike programming languages are called “high-level” languages. However, most people still require some training to use higher-level languages. This is particularly true of procedural languages.

**Procedural languages,** also known as **3GLs (third-generation languages),** are designed to express the logic – the procedures – that can solve general problems. Procedural languages, then, are intended to solve general problems. C++ is a procedural language. Consider the following C++ statement from a program that assigns letter grades based on the score of an exam. **if (score > = 90) grade = ‘A’;**

This statement tests whether the score is greater than or equal to 90. If it is, then the letter grade of **A** is assigned.

Like assembly languages, procedural languages must be translated into machine language so that the computer processes them. Depending on the language, this translation is performed by either a *compiler* or an *interpreter.*

• A **compiler** converts the programmer’s procedural language program, called the **source code,** into a machine language code, called the **object code.** This object code can then be saved and run later. Examples of procedural languages using compilers are the standard versions of Pascal, COBOL, and FORTRAN.

• An **interpreter** converts the procedural language one statement at a time into machine code just before it is to be executed. No object code is saved. An example of a procedural language using an interpreter is the standard version of BASIC. What is the difference between using a compiler and using an interpreter? When a program is run, the compiler requires two steps. The first step is to convert the entire program’s source code to object code. The second step is to run the object code. The interpreter, in contrast, converts and runs the program one line at a time. The advantage of a compiler language is that once the object code has been obtained, the program executes faster. The advantage of an interpreter language is that programs are easier to develop.

**Task-Oriented Languages: The Fourth Generation**

Third-generation languages are valuable, but they require training in programming. Task-oriented languages, also known as **4GLs (fourth-generation languages)** and **very high-level languages,** require little special training on the part of the user.

Unlike general-purpose languages, **task-oriented languages** are designed to solve specific problems. While 3GLs focus on procedures and how logic can be combined to solve a variety of problems, 4GLs are nonprocedural and focus on specifying the specific tasks the program is to accomplish. 4GLs are more Englishlike, easier to program, and widely used by nonprogrammers. Some of these fourth-generation languages are used for very specific applications. For example, **IFPS (interactive financial planning system)** is used to develop financial models. Many 4GLs are part of a database management system. 4GLs include query languages and application generators:

• **Query languages: Query languages** enable nonprogrammers to use certain easily understood commands to search and generate reports from a database. One of the most widely used query languages is SQL (structured query language). For example, let’s say that Advantage Advertising has a database containing all customer calls for service and that their management would like a listing of all clients who incurred overtime charges. The SQL command to create this list is **SELECT client FROM dailyLog WHERE serviceEnd > 17**

This SQL statement selects or identifies all clients (a field name from the dailyLog table) that required service after 17 (military time for 5:00 P.M.). Microsoft Access can generate SQL commands like this one by using its Query wizard.

• **Application generators:** An **application generator** or a **program coder** is a program that provides modules of prewritten code. When using an application generator, a programmer can quickly create a program by referencing the module(s) that performs certain tasks. This greatly reduces the time to create an application. For example, Access has a report generation application and a Report wizard for creating a variety of different types of reports using database information.

**Problem and Constraint Languages: The Fifth Generation**

As they have evolved through the generations, computer languages have become more humanlike. Clearly, the fourth-generation query languages using commands that include words like SELECT, FROM, and WHERE are much more humanlike than the 0s and 1s of machine language. However, 4GLs are still a long way from the natural languages such as English and Spanish that people use.

The standard definition of a **fifth-generation language (5GL)** is a computer language that incorporates the concepts of artificial intelligence to allow a person to provide a system with a problem and some constraints, and then request a solution. Additionally, these languages would enable a computer to *learn* and to *apply* new information as people do. Rather than coding by keying in specific commands, we would communicate more directly to a computer using **natural languages.**

Consider the following natural language statement that might appear in a 5GL program for recommending medical treatment.

**Get patientDiagnosis from patientSymptoms “sneezing”, “coughing”, “aching”**

**CONCEPT CHECK**

1. What distinguishes a lower-level language from a higher-level language?

2. Outline the five generations of programming languages.