Five Generations of Programming Languages

Machine, Assembly, High-Level, 4GL, and Natural Languages.

Programming languages have evolved in generations. Each new generation resulted in an improvement in the new languages that made it easier for people (programmers) to use them. The goal of a programming language is to communicate with the computer. The problem is that a computer, being an electro-mechanical device, can only understand a series of 2 state electrical signals (represented as: 1/0, high/low, on/off, true/false, 0volts/5volts, etc.). This section discusses how programmers are able to write code which a computer understands, without having to write it in its native "machine code".

This section of lecture notes discusses 5 generations of programming languages as listed below, and described in more detail in these lecture notes:

1. First generation 1945 Machine language (1's and 0's).
2. Second generation: 1950's Assembly languages (low-level instructions)
3. Third generation: 1960's High-level languages (COBOL, C, BASIC...)
4. Fourth generation: 1970's Very-high-level languages (NOMAD, FOCUS)
5. Fifth generation: 1980's Natural languages

First Generation: Machine Language

First generation languages were written in machine language. Machine language is a series of 0's and 1's. Although this is fine for computers, it is very difficult for programmers to comprehend. It wasn't too bad on very small computers, with a small set of machine instructions, but as computers grew in size and performance it became impractical to program using machine languages. Also, machine language is written for a particular machine, and a particular microprocessor, so machine code written for one computer cannot run on another computer. In other words, machine language is not portable. Although we no longer write programs in machine language, computers still require that a program be in machine language for the computer to execute it.

Below is some sample code written in Machine language for a particular computer. It will multiply 2 numbers together and store the product in a hardware register:

```
11110010  01110011  11010010  00010000  01110000  00101011
11100000  10010001  10000100  10000111  10010000  01010101
10100001  00001111  01010010  10010010  10000100  00100011
10010000  10001000  10000100  00001000  11001100  00100100
```

Second Generation: Assembly Language

Second generation languages are characterized by the use of symbolic instructions rather than 0's and 1's. The symbolic instructions are mnemonic (e.g., load, sum,),

which make them easier to learn and remember than machine language. Each instruction in assembly language corresponds to a machine operation Although assembly language gives programmers great control over the hardware, it is costly in terms of programmer time; it is also difficult to read, debug, and learn. Assembly language is used primarily today in some parts of system software. Like machine language, assembly language is designed for a specific machine and specific microprocessors, and is not portable.

Assembly language programs require a translator to convert the assembly language code into machine language, so the computer can execute it. These translators are known as **assemblers**.

Below illustrates some sample code written in assembly language. It too is written for a particular machine, and will multiply 2 numbers together and store the result in some hardware register.

```
PACK 210(8,13), 02B(4,7)
PACK 218(8,13), 02F(4,7)
MP      212(6,13), 21D(3,13)
SRP     213(5,13), 03E(0),5
UNPK 050(5,7),  214(4,13)
OI      054(7),X'FO'
```

Third Generation: High-Level Languages

Well, luckily for you, we will NOT be coding in any 1st or 2nd generation languages in this course. We will however, be coding in "C", which is considered a third generation, or high-level language. Third generation languages became more "English-like" and were much easier to write and debug than assembly language. Different third generation languages were written for different specialties. Some are better for business applications (COBOL), some for scientific and mathematical applications (FORTRAN, C), etc. Like assembly language, each third generation language requires a translator to convert the English-like, high-level code (**source code**) to machine code (**executable code**). Translators for third generation languages are called **compilers**. A few third generation languages use **interpreters** instead of compilers. A compiler converts every instruction in the program to machine language before any instruction is executed. Interpreters convert a single instruction, then executes it. This process is repeated for every instruction in a program. A program written in some high-level languages (such as "C"), can be compiled on one machine and executed on that machine. The source code can then be ported (copied) to a different computer, compiled on that different computer, and executed. High-level language code is considered to be portable across different hardware platforms. Some high-level languages include: FORTRAN (1954), COBOL (1960), BASIC (1965), Pascal (1970), C (1972), Ada (1979), C++ (1985).

Below is some sample code written in a variety of high-level languages. It represents the above code which is the multiplication of 2 numbers, and storing the product in some hardware location (in this case, in a variable in memory):

COBOL: MULTIPLY HOURS-WORKED BY PAY-RATE GIVING GROSS-PAY ROUNDED

C: gross_pay = hours_worked * pay_rate;

PASCAL: gross_pay := hours_worked * pay_rate;

Basic: g = h * p

As you can see above, the difference between one high-level language and another is something we call "Syntax". That is, how each instruction is represented so that the compiler can recognize it as a valid executable instruction.

Fourth Generation: Very-High-Level Languages

First, second and third generation languages are all classified as procedural languages. In a program written in a procedural language, not only must each instruction be coded correctly, each instruction must be in the proper sequence. Fourth generation languages (4GL) are non procedural. All you need to do is "tell" the program what you want it to do, and the language will create the program for you. Some categories of fourth-generation languages include:

- report generators (report write)
- query languages (SQL)
- application generators (*visual* whatevers)

I have limited experience with languages in this category. So, this is about all I have to

offer here. (Besides, "real" programmers code in 3rd generation languages!)

Fifth Generation: Natural Languages

Allows users to communicate with the computer using conversational commands that resemble human speech. Natural language development is part of the study of artificial intelligence. It allows commands to be framed in a more conversational way. (Huh?

Again, "real" programmers ...)