Language Translation, History

CS152

Chris Pollett

Sep. 3, 2008.

Outline

- Language Definition, Translation
- History of Programming Languages

Language Definition

- There are several different ways one can define a programming language. One way is to write a compiler and say that the language is specified by how the compiler outputs your code. This is called creating a **reference implementation**. Some languages like PHP, Perl only do this.
- Another way to create a standard specification for the language is some English (or other human language) **reference manual**. To do this one needs to specify two things:
  - **Language Syntax**: this describes what strings constitute programs in the given languages. A spec for the syntax might be to give a context free grammar. Such a grammar might have rules like:

    <if-statement> ::== if(<expression>) <statement> [else <statement>]

    String like "if' and 'else' in the above are called **tokens**. Their description form the **lexical structure** of the language.
  - **Language Semantics**: a specification of this must describe how a given syntactic structure should be implemented on the computer. This can be English, or using one of a few common formal semantics such as: **operational semantics, denotational semantics, or axiomatic semantics.**

Language Translation

- To be useful a programming language needs to have a **translator.**
- This is a program which either translates instances of the language into machine code, which can then be executes **(compiler);** or which takes instances of the languages and executes them as it read them **(interpreter).**
- You can also compile to an intermediate representation (for instance, a byte code compiler) and then have an interpreter for this representation **(pseudointerpreters).** Pascal, Java, Perl, PHP, Prolog, etc do this.
- A language is typically defined differently from the results of a particular translator. For instance, gcc does not completely implement the C99 spec.

More on Translation

- Translation is typically done by using a lexical analyzer which then sends tokens to a syntax analyzer (parser).
- One might also have a preprocessor that does an initial pass over the file to get it into a form suitable for the translator.
- A **runtime environment** for program data must also be generated (compiler) /maintained (interpreter).
- One might have several **passes** (I.e., readings of the file to compile) over the program to perform an entire compilation. You might also select for certain optimizations to be carried out during a given pass.
- Translation usually supports an error reporting/ debugging mechanism.
- The translator might also make use of pragmas in the code to control compiler options as compilation is done.

History of Programming Languages

My intention is to give a flavor of when various language constructs appeared rather than be complete.

- Pre-history
  - Many ancient cultures used natural languages to specify algorithms for computing things like volumes, areas, etc. Once committed to paper such an algorithm could be read and executed on another human.
  - Hero of Alexander (1st century AD) created automata for theaters. The controls for these automata could be "programmed" by winding ropes around their drive mechanisms in different ways.
  - In the 1800s, Joseph Marie Jacquard used punch cards to program how looms would weave things like carpets and tapestries.
  - This influenced Charles Babbage who also in the 1800s worked on very early mechanical computers such as the Analytical Engine which were never fully completed.

First Real Programming Languages

- FORTRAN - short for Formula Translation. Developed at IBM by John Backus 1954-1957. It was a procedural language. It introduced variables, arrays, loops, if statements. The goal was to produce code that was very close to human coded machine code in speed. Compilers for it are still among the fastest. Fortran 77 did not have a runtime stack (Fortran 90 did). So when you called a function or a subroutine, the values of the call parameters and local variables were what they were from the previous call.

Other Procedural languages of the Late 1950's

- Other procedural languages which came out a few years later were:
  - COBOL (1959-1960) -- designed for businesses. So code used lots of English tokens (an idea pioneered by Grace Hopper, US Navy) to try to make it readable by non-programmers. Language was first to have a way for programmers to define new data types via records. It also had a way (pictures) of creating sophisticated formatted output. Similar, formatted output can be seen in Perl.
  - Algol (1958-1960) -- first language to have its syntax specified using Backus-Naur form (BNF). It also allowed free format for the code rather than a fixed format based on columns on a punch card like Fortran and Cobol. It had a runtime stack supported recursion, introduced begin-end blocks, etc. It influenced many later languages like Pascal, C, Ada.

LISP

- Specified in a paper by John McCarthy in 1958.
- An interpreter appeared shortly thereafter. The first compiler appeared in 1962.
- It was the first functional programming language.
- It was untyped and based on a generalized notion of list.
- It supported recursion.
- It was the first language to have garbage collection.
- One of the most popular languages in AI research.

Languages of 1960s

- PL/I (1963-1964) Developed at IBM for the IBM 360. It supported concurrency and exception handling.
- SNOBOL (1962-1967) R. Griswald, Bells Labs. First language to support a wide set of string manipulation features. Ideas later incorporated in languages like sed, awk, Perl.
- Simula 67 (1965-1967) Developed by Nyqaard and Dahl at the Norwegian Computing Center. Introduced the notion of a class. First object-oriented language.
- BASIC (1964) Kenemy and Kurtz, Dartmouth College. Perhaps, the first language geared toward teaching programming.

Languages of the 1970s

- Pascal - (1971) Developed as an educational language by N. Wirth in reaction to how complicated Algol68 was. It was used to program the OS of early Mac computers.
- C - (1972) Developed by Denis Ritchie Bell Labs. Used to write the Unix OS. Tried to be convenient for programmers to write (avoiding unnecessary syntax) and to be close to the underlying machine. (mid-level language)
- CLU - (1974-1977) OO language developed by Barbara Liskov, MIT. Introduced the notion of **iterator** to OO languages. Also, had very good exception handling.
- Prolog - (1972) developed in Marseille, France by A. Colmerauer. Most popular logic programming language. Used in AI, web intelligence, etc. Influenced Datalog and Query By Example database access.

Languages of the 1980s

- Ada - developed by the Department of Defence. Introduced the notions of **packages** for ADTs and **tasks** for concurrency. My uncle (Whittaker) involved in its design.
- Smalltalk - (1980) OO language developed at Xerox Parc. Used in first GUI OS. Many OO design patterns were first described for this langauge.
- Scheme - (1975-1978) simplified version of LISP developed at MIT for initially educational purposes. It became with the publication of Abelson Sussman 1985.
- ML - (1978-) Milnor, at Edinburgh. Typed functional programming language initially used for theorem proving (HOL).
- C++ - (1983) OO language developed by Bjarne Stroustrup, AT&T. First widely popular OO language
- Perl - general purpose, scripting, string manipulation language developed by Larry Wall. One of the First, language used for CGI programming.

Languages of the 1990s to Present

- Java - (1994-1995) developed by Gosling at Sun. Simplified (compared to C++) C-like OO language with a large initial library of classes. Popularized byte-code compilers. Used both client-side and server-side for web development. Initially, designed for embedded devices.
- Javascript - (1994-1995) developed by Eich at Netscape. Object-based. Introduced concept of Document Object Model.
- PHP - server-side templating language. Developed by Rasmus Lerdoff. One of the most popular web programming languages.
- Many other recent web languages Python, Ruby, etc. Also lots of document languages based on XML.