Lecture 12: Procedural and Functional Languages

Most of the course has focused on procedural (also called imperative) languages, object-oriented languages, and hybrid languages. This lecture sheds some more light on procedural languages, and discuses functional languages.

- Summary of Imperative Languages
- Introduction to Functional Languages
- Fundamentals of Functional Languages
- Summary of Popular functional Languages

12.1 Summary of Imperative Languages

Imperative (also called procedural) languages are among the oldest forms of programming languages. Procedural languages trace back to Fortran. And C is clearly the most prominent of these languages. Here is a summary of both languages:

Fortran
- Over 50 years old (first introduced in 1957)
- Suited for the scientific environment
- Inelegant but very efficient
- Fortran 90 adds modern data and control features to give it the power of languages like Pascal and C

C
- Over 30 years old (first introduced in 1971)
- Developed from CPL, BCPL, and ALGOL 68
- Compact syntax and efficient execution
- Somewhat cryptic
- Ideal for systems programming (most embedded systems, DBMS suites, operating systems, and communications systems have been written in C)

12.2 Introduction to Functional Languages

Functional programming grew out of a need to support mathematical functions. This later expanded into the field of artificial intelligence (AI). Traditionally, AI was a branch of computer science that represented the confluence of interests from linguistics (mainly natural language processing), psychology (mainly modeling information for storage and retrieval), and mathematics (mainly theorem proving). Today, AI still includes these areas, but has expanded into other areas such as robotics, expert systems, neurology, and biometrics, among others.

LISP remains the most prominent functional language. Other functional languages include CLOS, ML, Haskell, and Scheme.

12.2.1 Simple Functions

A simple function is specified as a function name, a list of parameters, and a mapping expression.

Example 1:

Consider the function called **Cube**, defined as
Cube(x) = x * x * x, where x is a real number
Then Cube(3) yields the result 27

Consider the functions **Parabola**, defined as
Parabola(x, a, b, c) = a * x * x + b * x + c, where x, a, b, c are real numbers
Then Parabola(2,1,2,3) = 1(2*2) + 2(2) + 3 = 11

12.2.2 Functional Forms

A functional form (or higher order function) can accept a function as a parameter, or yield a function as its result. This includes composite functions, *apply-to-all* function forms (denoted by α), and several others.

Example 2:

Let $f(x) = x^3$ and $g(x) = x^2 + 2$
Then $f(g(x)) = (x^2 + 2)^3$
And $g(f(x)) = (x^3)^2 + 2$

Also assuming f and g as defined above
$\alpha(g(0,1)) = (0^2 + 2, 1^2 + 2) = (2,3)$
$\alpha(f(0,1)) = (0^3, 1^3) = (0,1)$

12.3 Fundamentals of Functional Languages

A functional language tries to emulate mathematical functions to the greatest level of accuracy. Functional languages differ from imperative, OO, and hybrid languages in the following way:

- Imperative languages achieve function implementation by allowing the programmer to implement algorithms by defining and manipulating variables. These variables are stored in specific memory locations during program execution.
- Functional languages achieve function implementation by allowing the programmer to define functions as primary units of manipulation.

It should be clear to you that functional programming languages are more suited for complex mathematical problem solving, and provide more flexibility than other programming languages in this particular problem domain.

Functional languages exhibit *referential transparency*: They do not require variable definitions or assignment statements. One simply defines functions. Repetition is typically replaced by recursion. A program is a collection of function definitions. Program execution is the application of the function(s) to the specific parameters provided. Moreover, the result is always consistent for the same parameters.

The functional language consists of a number of primitive functions and functional forms that can be used by the programmer to develop more complex functions.

12.4 Summary of Popular Functional Languages

12.4.1 LISP

LISP supports two data types, namely atoms and lists. Nested lists are supported.

See [Sebesta 2012]

12.4.2 Scheme

A dialect of LIST.

See [Sebesta 2012]

12.4.3 Common LISP

An attempt to unite different dialects of LISP.

12.4.4 ML

- Strongly typed, whereas Scheme is type-less
- Supports abstract data types, and includes an exception handler
- Seems more useful than LISP and Scheme

See [Sebesta 2012]

12.4.5 Haskell

Similar to ML

See [Sebesta 2012]

12.5 Summary and Concluding Remarks

Here is a summary of what we have covered in this brief lecture:

- Procedural languages are among the oldest forms of programming languages, tracing back to back to Fortran of the 1950s.
- Functional programming is geared towards supporting mathematical functions and artificial intelligence (AI). Specific branches of mathematics and AI that are easily facilitated include linguistics (mainly natural language processing), psychology (mainly modeling information for storage and retrieval), mathematical theorem-proving, robotics, expert systems, neurology, and biometrics.
- Functional languages exhibit *referential transparency*: They do not require variable definitions or assignment statements. Rather, their building blocks are functions. Repetition is replaced by recursion. A program is a collection of function definitions. Program execution is the application of the function(s) to the specific parameters provided. Moreover, the result is always consistent for the same parameters.
- Among the prominent functional languages are LISP, Scheme, ML, and Haskell.

12.5 Summary and Concluding Remarks (continued)

Functional programming is rather interesting; you are strongly encouraged to take some time and familiarize yourself with at least one of the languages mentioned in this lecture. The next section provides links to online resources for each of the languages mentioned above. The next scheduled topic discusses logic programming languages.

12.6 Recommended Readings

[Dybvig 2009] Dybvig, R. Kent. 2009. *The Scheme Programming Language* 4th Edition. Cambridge MA: MIT Press. Accessed on December 26, 2014. http://www.scheme.com/tspl4/

[Haskell 2013] Haskell.org. 2013. "The Haskell Programming Language." Accessed on December 26, 2014. https://www.haskell.org/haskellwiki/Haskell

[Sebesta 2012] Sebesta, Robert W. 2012. *Concepts of Programming Languages* 10th Edition. Colorado Springs, Colorado: Pearson. See chapter 15.

[SML/NJ 2013] SML/NJ. 2013. "Standard ML of New Jersey." Accessed on December 26, 2014. http://www.smlnj.org/

[Tutorialspoint 2013] Tutorialspoint. 2013. "LISP Programming." Accessed on December 26, 2014. http://www.tutorialspoint.com/lisp/