Principles of Programming Languages(CS F301)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Introduction to Functional Programming(Ch.16 of T1)

BITS Pilani
Hyderabad Campus

Prof R Gururaj

Introduction to Functional Programming (Ch.16)

Functional Programming paradigm is based on mathematical functions, and is the design basis of the most important non-imperative languages.

In 1978, during his ACM award-lecture, *John Backus*, made a case that purely functional languages are better than imperative languages because they result in programs that are more readable, more reliable and more likely to be correct.

The crux of his argument was – purely functional languages are easy to understand both during development and after, because the meaning of their expressions are independent of the context.

John Backus proposed the language *Functional Programming* (FP). But did not succeed.

The languages ML, Haskell, F# gained some popularity.

One fundamental characteristic of programs written in imperative languages is that the programs have a *state* represented by the values assigned to its variables.

The program state keeps changing through the execution of the program.

All readers of the program understand the use of its variables and how the program's state changes through execution. For larger programs this becomes a very difficult task. But Functional programming does not have this issue since they don't have variables concept.

LISP began as pure functional language.

But soon acquired some important features of imperative languages, that increase its efficiency.

Still LISP is the only widely used Functional Programming Language.

LISP dominates the area of knowledge representation, machine learning, intelligent training systems and modeling speech.

Common LISP is amalgam of early dialects of LISP. (1980)

Scheme is small static scoped dialect of LISP.

Scheme is popularly used for teaching Functional languages.

Some Universities use Scheme to teach introduction to programming.

The development of the typed functional languages primarily ML, Haskell, OCml and F# has led to significant expansion of the areas of computing in which Functional languages are now used in areas like- Database Processing, Financial modeling, Statistical analysis and bio-informatics.

Mathematical functions

A mathematical function is a mapping of members of one set called *domain* set to another set called *range* set.

A function definition specifies the domain and range sets, either explicitly or implicitly along with the mapping.

The mapping is described by an expression or in some cases by a table.

A mathematical function takes an element from domain set as a parameter and yields an element of range set.

A function maps its parameter to a value or values, rather than specifying the sequence of operations.

Ex: $cube(x) \equiv x*x*x$

Here in this function definition the domain and range both are real numbers.

Lambda (λ) calculus is inspiration for functional programming languages.

The lambda(λ) notation focus on the tasks a function performs than on name.

It can be understood as defining nameless functions.

*λ(x)x*x*x* is a lambda expression.

Ex:

Application of lambda expression

*(λ(x)x*x*x) (2)* results in 8.

The formal computational model that uses lambda expression is known as lambda calculus.

Fundamentals of Functional programming languages.

1. The objective of the design of a functional programming languages is to mimic mathematical functions to the extent possible.
2. This results in a approach to problem solving different than approach used by imperative languages.
3. A purely functional language does not use variables or assignment statements, thus freeing the programmer from the concerns related to memory cell, or state of the program.
4. Without variables iterations not possible. But can be specified with recursion.
5. The execution of a function always gives same results when the parameter is same. This is called *referential transparency.*

The execution of a function always gives same results when the parameter is same. This is called *referential transparency*.

This makes testing easier because each function can be tested without any concern for its context.

Haskell is pure Functional Language.

But many other Functional languages include some features taken from imperative languages.

Functional languages are mainly interpreted.

But in the recent past compilation is also there.

LISP

Very first Functional language.

Introduced in 1959 at MIT by John McCarthy.

For Functional paradigm, it like how FORTRAN is for Imperative paradigm.

There are two categories of Data objects in original LISP.

- Atoms: character strings or numeric constants.
- Lists: list element are pairs.

First part of a list is data which is a pointer to either atom or another List.

The second part of a list is a pointer to next element or empty.

Elements are linked together in lists with second part.

Lists are specified in LISP by delimiting their elements with parenthesis.

The elements of simple list are restricted to atoms.

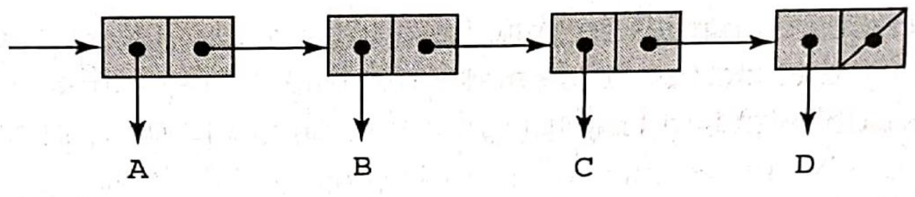
(A B C D)

Nested lists are possible.

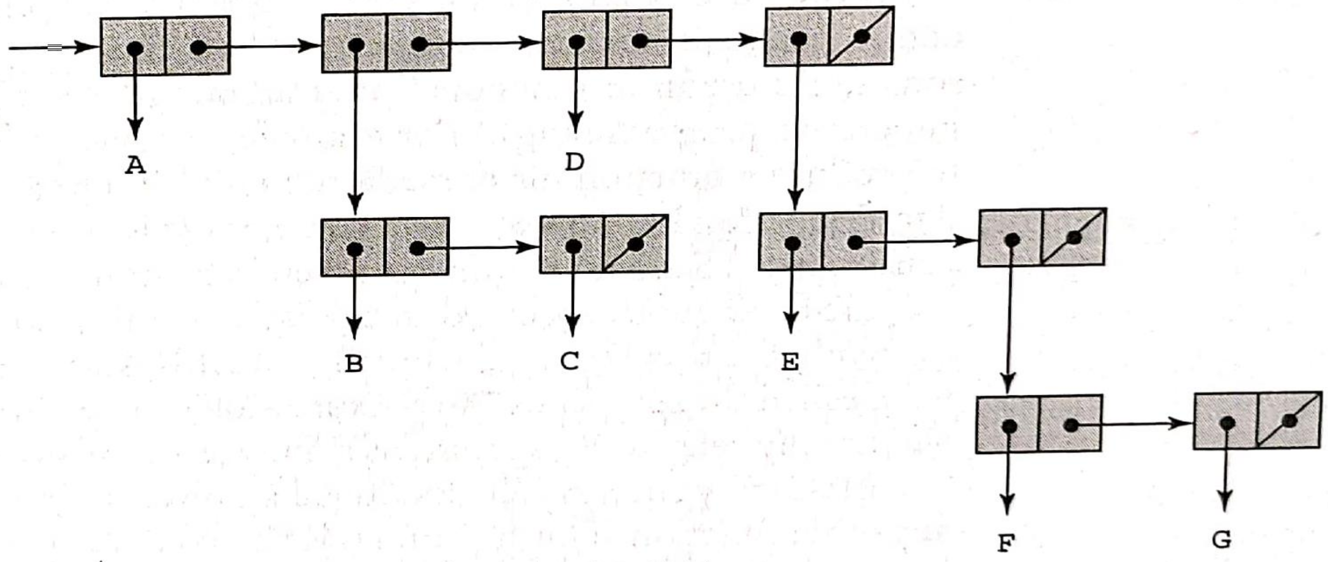
(A (B C) D (E (F G)))

Figure 1

Internal representation of two LISP lists



(A B C D)



(A (B C) D (E (F G)))

Function definition and function call also is represented n List notation.

(+ 5 7) =12

(+ 3 4 7 6) = 20

Other Functional languages

Scheme is a dialect of LISP.

A scheme program is collection of function definitions.

Lambda expressions are used.

Common LISP is an amalgam of LISP dialects – 1980.

It allows both astatic and dynamic typed variables and includes many imperative features.

ML is static scoped and strongly typed functional language., used syntax close to imperative languages. Includes exception handling, variety of data structures and abstract data types.

Haskell is similar to ML. Unlike Scheme and ML, Haskell is pure FL.

F# is a .NET programming language that supports functional, imperative, an Object Oriented programming.

It can interoperate with other .NET languages and has access to the .NET class library.

Summary of Ch.16 (Functional Programming.)

1. Introduction
2. Mathematical Functions
3. Fundamentals of Functional Programming
4. Introduction to LISP
5. Other Functional Languages.