

函数式编程原理

课程报告

**姓 名： 汪宇飞**

**学 号： U202015375**

**班 级： CS2003班**

**指导教师： 郑然**

**计算机科学与技术学院**

**2021 年12月 27日**

# 函数式语言家族成员调研

根据百度百科的定义与介绍，函数式语言(functional language)一类程序设计语言，是一种非冯·诺伊曼式的程序设计语言。函数式语言主要成分是原始函数、定义函数和函数型。这种语言具有较强的组织数据结构的能力，可以把某一数据结构(如数组)作为单一值处理；可以把函数作为参数，其结果也可为函数，这种定义的函数称为高阶函数，程序就是函数，程序作用在结构型数据上，产生结构型结果，从根本上改变了冯·诺伊曼式语言的“逐词”工作方式。

最早的函数式编程语言是麻省理工学院的人工智能研究先驱McCarthy J于1958年提出的LISP语言，它是如今大部分函数式编程语言的起源，其衍生出的语言众多，主要家族结构如图1-1所示。



图1-1 函数式语言家族成员结构

以下对家族中成员逐一介绍。

## LISP

LISP语言于1958年由麻省理工学院的人工智能研究先驱McCarthy J提出，是最早的函数式语言。它的特征是表处理语言，适合于数学计算，基于λ演算所创造，采用抽象数据列表与递归作符号演算来衍生人工智能。

LISP为函数式程序设计语言，所有运算都能以函数作用于参数的方式来实现。它没有命令式语言程序中常见赋值语句和变量，重复的过程可以使用递归的函数调用来表示，并不需要使用循环模式。

LISP是古老的函数语言、弱类型、动态推断，其代码本身就是各种列表。每一个表项均可以储存任何类型的数据如数字、函数、符号或一个子表等，在编码时，可以随时操作以更新列表。

LISP核心的操作符只有7个操作符：quote、atom、eq、car、cdr、cons、cond。前三者quote、atom、eq用于符号的推断；car、cdr、cons操纵表格；cond负责分支判断。这种简洁定义，非常接近图灵机原型的纯函数式语言，是现代语言完全无法比拟的。

LISP作为弱类型这优点相对缺点则是运行效率的低下。原始定义简洁的缺点使到大型开发工程变得困难，自底层到高层，自二维表查询到面向对象，使用者需要嵌入更多的函数来实现，致使LISP众多方言的衍生。



图1-2 LISP语言的提出者 John McCarthy

## COMMON LISP

Common Lisp，缩写为 CL（不要和缩写同为CL的组合逻辑混淆），是Lisp的众多方言之一，与Scheme合称现代两大Lisp方言，标准由ANSI X3.226-1994定义。

Common Lisp是一个现代的，多重范式的，高性能，可编译的标准化ANSI编程语言。相对于各种嵌入在特定产品中的Lisp方言，例如Emacs Lisp 和 AutoLISP，Common Lisp 是一种通用用途的编程语言。不像很多早期的Lisp，Common Lisp同Scheme一样，变量有作用域。

Common Lisp具有高度灵活性，对象化编程以及快速的框架能力提供优异的支持。同时它还提供强大的宏能力，使编程者在编码时定制自己的应用，并给编程者提供了高度灵活的运行环境，可在运行时修改和调试程序。其多重范式语言特性还允许编程者选择适当的方法和范式适应编写的应用程序。

## Scheme

Scheme语言于1975年由麻省理工大学的Gerald J. Sussman 和 Guy L. Steele Jr.创造。它遵循极简主义哲学，以一个小型语言核心作为标准，加上各种强力语言工具（语法糖）来扩展语言本身。

MIT曾用Scheme作为计算机系入门课程的编程语言。计算机程序语言界著名的魔法书《计算机程序的构造和解释》（又称SICP）正是利用Scheme来解释程序设计。

历史悠久的Scheme依然活跃，拥有针对各种计算机平台和环境的实现，例如Racket、Guile、MIT Scheme、Chez Scheme等。Guile是GNU工具体系里最重要的部件之一，被许多自由软件和开源软件作为内置脚本语言使用。

## Clojure

Clojure是一种动态的、强类型的、寄居在JVM上的语言。其特性包括：

1.函数式编程基础，包括一套性能可以和典型可变数据结构媲美的持久性数据结构

2.由JVM提供的成熟的、高效的运行时环境：所以Clojure可以使用Java类库，反之Clojure库也可以被Java使用

3.跟JVM/Java的互操作能力使得很多架构、运维方面的需求可以得到满足：Clojure代码可以像Java代码一样被打包，然后部署到任何Java应用可以部署的地方

4.一套提供并发、并行语义的机制：Clojure的应用类型强制我们把对象的状态和对象的标识区分开，对于多线程的支持使得我们不用手动加锁，解锁也能编写多线程代码

5.是一种Lisp方言，因此提供了非常灵活、强大的元编程能力：Clojure保留了Lisp的最好的特性，去掉了Lisp方言的缺陷。



图1-3 Clojure语言的logo

## 5.ML

ML语言于1979年由Milner R提出，其特征是非纯函数式编程语言，强类型，允许副作用和指令式编程，其语法是从ISWIM中获得的灵感。

ML语言有Standard ML（SML）和Caml这两个较大分支，其中SML即我们这一课程使用的语言，Caml具体由OCaml语言实现，在下一小节中将介绍。

ML特性有惰性求值的求值策略，一阶类型函数， 带有垃圾收集的自动内存管理， 参数多态，静态数据类型，类型推断，代数数据类型，模式匹配和异常处理。

不像Haskell，ML使用表达式求值，也就是说所有的子表达式总是被求值。导致的一个结果是你不能使用无穷表。然而，惰性求值产生的无穷表可以通过使用匿名函数来模拟。

ML可以算一种具备命令式语言特点的函数型语言，或者说面向函数的命令型语言。和Lisp一样，ML具有非常灵活的函数功能。例如一个表达式的值可能就是一个函数，这个函数可以被作为参数传递给另一个函数，或者函数的返回值就是一个函数。同时和Algol类的语言比较接近的是，ML的语法象命令型的，而且用起来象用Algol家族的很多比较新的后代们一样方便。而且ML有并行扩展，可以用来写并行系统；甚至还有面向对象扩展。

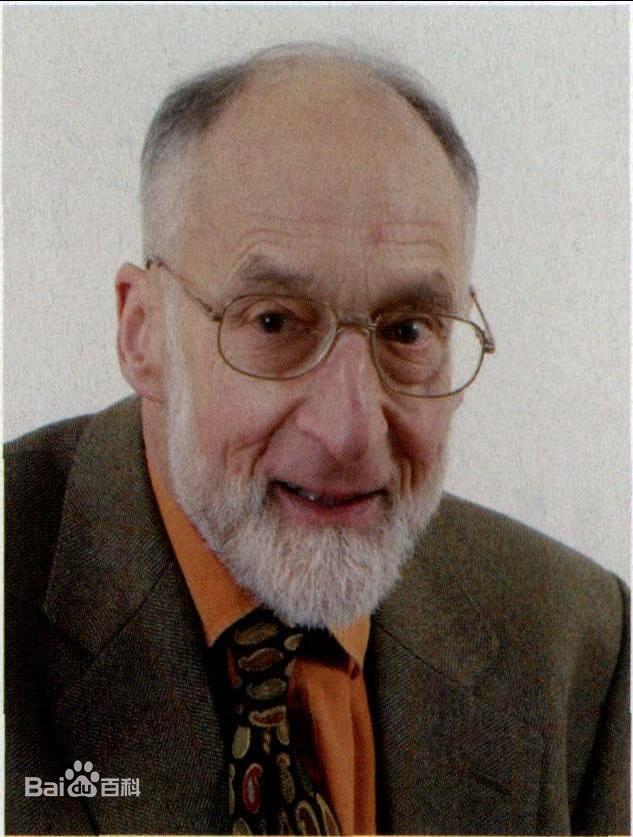


图1-4 ML语言的提出者Milner R

## 6.OCaml

OCaml语言于1996年由由Xavier Leroy，Jérôme Vouillon，Damien Doligez，Didier Rémy及其他人于1996年创立，是一个开源项目，是Objective Caml的简称。其特征为非函数式编程语言，强静态类型。

OCaml将Caml语言在面向对象方面做了延展。Caml 是函数式编程语言，它的扩展语言还有基于微软.net平台的 f# (fsharp)语言。Caml 的代码大多可以在f#中使用。F#的开发工具有VS .net，Caml的代码也可使用。

OCaml的开发工具包含交互式顶层解释器，字节码编译器，以及最优本地代码编译器。OCaml有一个巨大并强悍的标准库，这使得她可以像Python或者Perl语言一样可以方便地开发各种应用程序，健壮的模块化与面向对象编程结构又使得她可以胜任大规模软件工程项目。

OCaml是Caml的继承者，CAML的缩写最初代表着Categorical Abstract Machine Language(分类抽象机语言)，不过后来OCaml将这个抽象机淘汰掉了。如今人们更喜欢把CAML理解为Collaborative Application Markup Language(协作应用程序标记语言)。

https://bkimg.cdn.bcebos.com/pic/4bed2e738bd4b31cfe2438be87d6277f9e2ff89f?x-bce-process=image/resize,m_lfit,w_268,limit_1/format,f_auto

图1-5 OCmal语言的骆驼logo

## 7.Miranda

Miranda语言于1985年由Turner D提出，其特征是基于ML的惰性纯函数式语言，属于强类型。其发表时配套发布了一个C语言编写的编译器，可以在Linux、Unix系统上运行，之后再1987年和1989年发布了更新。

## 8.Haskell

Haskell语言于1988年由Hudak P和Wadler P提出，其特征是纯函数式语言，支持惰性求值、高阶、强类型、模式匹配、列表内包、类型类和多态。是一种标准化的，通用的纯函数编程语言，有非限定性语义和强静态类型。

在Haskell中，“函数是第一类对象”。作为一门函数编程语言，主要控制结构是函数。Haskell语言是1990年在编程语言Miranda的基础上标准化的，并且以λ演算为基础发展而来。这也是为什么Haskell语言以希腊字母“λ”（Lambda）作为自己的标志。Haskell具有“证明即程序、命题为类型”的特征。



图1-6 Haskell语言的logo

## 9.F#

F#语言于2002年由微软提出，其特征是基于OCaml的非纯函数式编程语言，强静态类型，适合程序核心数据多线程处理。

F#自2002年开始研发，2005年发布了第一个版本，2007年底正式从研发专案转移至产品部门，并决定将F#置入Visual Studio 2010。截止目前最新的F#为F# 4.0。搭载于Visual Studio 2015中。

微软计划将慢慢整合F#至.NET平台并使F#成为.NET平台的计算辅助语言。

众所周知，F#是一种函数型程序设计语言。然而F#对IP与OOP的支持几乎一样的出色。F#也许终将成为程序核心数据多线程处理的首选，而C#与VB等将在用户界面交互设计方面继续发挥其强大的潜力。

以目前来看，随着FP在程序设计中的重要性日渐凸显，F#身为微软唯一的FP语言，其位置特殊，容易引起关注。对一部分人来说，这语言所带来的一些特性以及其对FP的特性的全面支持，可能会带来一次大的革变。

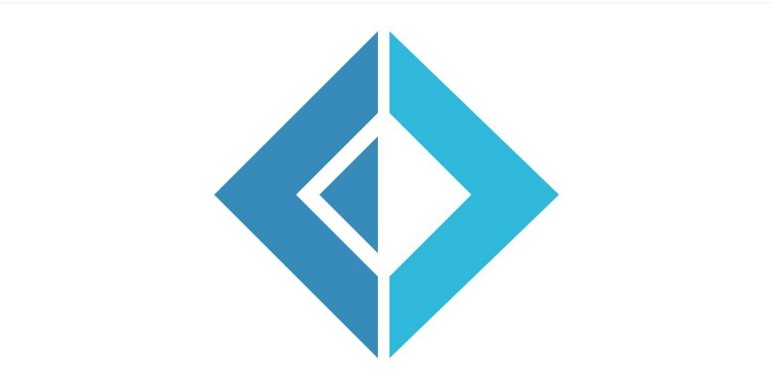


图1-7 F#语言的logo

除上述函数式编程语言之外，还有以下几种函数式编程语言：

## 10.Erlang

Erlang语言于1987年由Erlang A K提出，其特征是多范式编程语言，支持函数式、并发式及分布式编程风格。在1991年由爱立信公司向用户推出了第一个版本，经过不断的改进完善和发展，在1996年爱立信又为所有的Erlang用户提供了一个非常实用且稳定的OTP软件库并在1998年发布了第一个开源版本。Erlang同时支持的操作系统有linux,windows,unix等，可以说适用于主流的操作系统上，尤其是它支持多核的特性非常适合多核CPU，而分布式特性也可以很好融合各种分布式集群。

Erlang是一种通用的面向并发的编程语言，它由瑞典电信设备制造商爱立信所辖的CS-Lab开发，目的是创造一种可以应对大规模并发活动的编程语言和运行环境。Erlang问世于1987年，经过十年的发展，于1998年发布开源版本。Erlang是运行于虚拟机的解释性语言，但是也包含有乌普萨拉大学高性能Erlang计划（HiPE）开发的本地代码编译器，自R11B-4版本开始，Erlang也开始支持脚本式解释器。在编程范型上，Erlang属于多重范型编程语言，涵盖函数式、并发式及分布式。顺序执行的Erlang是一个及早求值,单次赋值和动态类型的函数式编程语言。

Erlang是一个结构化，动态类型编程语言，内建并行计算支持。最初是由爱立信专门为通信应用设计的，比如控制交换机或者变换协议等，因此非常适 合于构建分布式，实时软并行计算系统。



图1-8 Erlang语言的logo

## 11.Scala

Scala语言于2001年由联邦理工学院洛桑的Odersky M提出，其特征是基于Funnel设计多范式编程语言，支持对象式和函数式编程的典型特征。

Scala是一种纯面向对象的语言，每一个值都是对象。对象的数据类型以及行为由类和特征描述。类抽象机制的扩展有两种途径。一种途径是子类继承，另一种途径是灵活的混入机制。这两种途径能避免多重继承的种种问题。

截至2009年9月，最新版本是版本2.7.6。Scala 2.8预计的特性包括重写的Scala类库、方法的命名参数和默认参数、包对象，以及Continuation.

2009年4月，Twitter宣布他们已经把大部分后端程序从Ruby迁移到Scala，其余部分也打算要迁移。此外， Wattzon已经公开宣称，其整个平台都已经是基于Scala基础设施编写的。



图1-9 Scla语言的logo

# 二、上机实验心得体会

我感受最深的实验是实验三中有关minheap树的几个相关函数的实验。虽然之前在数据结构课程中已经学习过minheap树相关的知识，了解过minheap树的性质并用C语言实现过它的相关功能，但这次用SML语言实现仍然有不一样的感觉，也加深了我对函数式编程的理解。

首先是实现函数的时候，由于函数是需要通过递归思想来实现的，所以要更加理解minheap树的性质，想明白某一步操作到底是如何实现的，对这一步操作理解透彻，然后才能弄清楚然后用递归思想来实现相关功能并保证编程的正确性。

其次是在实现相关功能的时候，一定要考虑细致，将可能出现的各种情况都要想到，比如列表为空的时候应该如何如何处理，或者参数为0的时候应该如何如何处理等一系列情况，均需要考虑并斟酌如何实现且保证考虑到的是正确的而不是臆想的，或者考虑到的某种情况是否已经被包含在其它情况中。只有细致周全的思考和斟酌才能得出正确完备并且简洁的好代码，成功地实现相关函数。

再就是由于这一功能较为复杂，在实现的函数中用到了let……in……end和case……of……这两个语句，通过完成这次实验我对这两个语句的功能与使用有了更深的认识。我觉得这两个语句是很实用的，在十分依赖递归的SML语言中，这两个语句的使用给了编程者更多的编程思路和想法，将一些可能仅仅靠递归难以实现的程序变得简单许多。

最后，在我看来函数式编程的思想对于我的编程能力的提升和编程思维的开导与启发上，是有着很大的作用的，虽然这门课学时不多，而且正如老师所说，我们只是被引进了函数式编程的大门，但是函数式编程的思想让我受益良多。它向我展示了如何在命令式编程之外，用一个个函数来实现各种功能的妙处和高效性。当然，命令式编程和函数式编程各有利弊，我们要做的就是取其精华去其糟粕，将这两种编程方式融会贯通，运用在我们的程序之中，让我们的编程在正确的同时高效简洁。总而言之十分感谢郑老师以及这门课程，让我见到了编程的更多可能性。

# 三、课程建议和意见

就我个人而言课程内容设置是比较合适的，课程内容量正好，不会很多也不会很少，难度也适中。唯一可能需要改变一下的是实验课对于函数的实现作业不足，可以适当多加一些题目供学生练习，以更好地掌握相关知识。