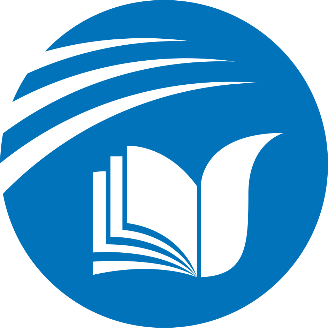
**ĐẠI HỌC THÁI NGUYÊN**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

------- 🙞🕮🙜 -------



**BÁO CÁO BÀI TẬP LỚN**

***HỌC PHẦN:* PHƯƠNG PHÁP LUẬN LẬP TRÌNH**

***ĐỀ TÀI:***  **“TÌM HIỂU VỀ LẬP TRÌNH HÀM”**

Nhóm sinh viên:

Dương Văn Sơn

Nguyễn Vũ Thành Tuyển

Trần Nhật Trường

Nguyễn Thanh Tuân

Lớp: Kỹ thuật phần mềm K15A

Giảng viên: ThS. Nguyễn Lan Oanh

*Thái Nguyên, tháng 3 năm 2019*

**MỤC LỤC**

[CHƯƠNG 1. TÌM HIỂU LẬP TRÌNH HÀM 3](#_Toc4402506)

[1.1. Tổng quan về lập trình hàm 3](#_Toc4402507)

[1.2. Hàm hạng nhất và hàm bậc cao 4](#_Toc4402508)

[1.3. Hàm thuần túy 5](#_Toc4402509)

[1.4. Đệ quy trong lập trình hàm 5](#_Toc4402510)

[1.5. Tính toán chặt và không chặt 6](#_Toc4402511)

[1.6. Kiểu, tính đa kiểu, kiểu dữ liệu đại số và so trùng mẫu 7](#_Toc4402512)

[1.7. Lập trình hàm trong ngôn ngữ phi hàm 8](#_Toc4402513)

[CHƯƠNG 2. GIỚI THIỆU VỀ NGÔN NGỮ LISP 9](#_Toc4402514)

[TÀI LIỆU THAM KHẢO 11](#_Toc4402515)

[PHÂN CÔNG & ĐÁNH GIÁ CÔNG VIỆC THÀNH VIÊN NHÓM 12](#_Toc4402516)

# TÌM HIỂU LẬP TRÌNH HÀM

## Tổng quan về lập trình hàm

Trong ngành khoa học máy tính, lập trình hàm là một mô hình lập trình xem việc tính toán là sự đánh giá các hàm toán học và tránh sử dụng trạng thái và các dữ liệu biến đổi. Lập trình hàm nhấn mạnh việc ứng dụng hàm số, trái với phong cách lập trình mệnh lệnh, nhấn mạnh vào sự thay đổi trạng thái. Lập trình hàm xuất phát từ phép tính lambda, một hệ thống hình thức được phát triển vào những năm 1930 để nghiên cứu định nghĩa hàm số, ứng dụng của hàm số, và đệ quy. Nhiều ngôn ngữ lập trình hàm có thể được xem là những cách phát triển giải tích lambda.

Trong thực tế, sự khác biệt giữa hàm số toán học và cách dùng từ "hàm" trong lập trình mệnh lệnh đó là các hàm mệnh lệnh có thể tạo ra hiệu ứng lề, làm thay đổi giá trị của một phép tính trước đó. Vì vậy các hàm kiểu này thiếu tính trong suốt tham chiếu, có nghĩa là cùng một biểu thức ngôn ngữ lại có thể tạo ra nhiều giá trị khác nhau vào các thời điểm khác nhau tùy thuộc vào trạng thái của chương trình đang thực thi. Ngược lại, trong lập trình hàm, giá trị xuất ra của một hàm chỉ phụ thuộc vào các tham số đầu vào của hàm, vì thế gọi hàm f hai lần với cùng giá trị tham số x sẽ cho ra cùng kết quả *f(x)*. Việc loại bỏ hiệu ứng lề có thể làm cho chương trình dễ hiểu hơn rất nhiều và người ta có dự đoán được hành vi của một chương trình, đó chính là một trong các động lực chính cho sự phát triển của lập trình hàm.

Các ngôn ngữ lập trình hàm, đặc biệt là các loại thuần lập trình hàm, có ảnh hưởng lớn trong giới học thuật hơn là dùng để phát triển các phần mềm thương mại. Tuy vậy, các ngôn ngữ lập trình hàm nổi bật như Scheme, Erlang, Objective Caml và Haskell đã được nhiều tổ chức khác nhau sử dụng trong các ứng dụng công nghiệp và thương mại. Lập trình hàm cũng được sử dụng trong công nghiệp thông qua các ngôn ngữ lập trình chuyên biệt như R (thống kê), Mathematica (toán học hình thức), J và K (phân tích tài chính), F# trong Microsoft.NET và XSLT (XML). Các ngôn ngữ chuyên biệt dạng khai báo được sử dụng rộng rãi hiện nay như SQL và Lex/Yacc, cũng sử dụng một số thành phần của lập trình hàm, đặc biệt để tránh các giá trị biến đổi. Các bảng tính (spreadsheet) cũng có thể được xem là các ngôn ngữ lập trình hàm.

## Hàm hạng nhất và hàm bậc cao

Trong Lập trình hàm (Functional Programming), một hàm sẽ được coi như là một hàm hạng nhất trong ngôn ngữ đó. Hay nói cách khác, hàm sẽ giống như các loại giá trị (số, text, object,...) khác.

Các đối tượng cơ sở được gọi là có bậc 0, các hàm có các tham đối là những đối tượng cơ sở được gọi là các hàm bậc 1. Một hàm bậc n nhận các tham đối bậc nhỏ hơn n, để có các biểu thức bậc cao hơn.

Trong các ngôn ngữ hàm, hàm luôn luôn được xử lý như những đối tượng được truyền tham đối để trả về kết quả và được lưu giữ trong các cấu trúc dữ liệu. Một hàm nhận một hàm khác như là một tham đối được gọi là hàm bậc cao. Các hàm bậc cao mang lại tính hiệu quả và là nền tảng (corner-stone) của lập trình hàm.

Hàm bậc cao (Higher-order Functions) là các hàm hoặc nhận hàm làm tham số, hoặc trả về hàm, hoặc vừa nhận hàm làm tham số vừa trả về hàm.

Các hàm bậc cao có liên hệ chặt chẽ với hàm hạng nhất, ở chỗ các hàm bậc cao và hàm hạng nhất đều cho phép nhận hàm số làm tham số và trả về các hàm khác. Sự khác biệt giữa hai loại này rất mờ nhạt: "bậc cao" mô tả một khái niệm hàm trong toán học tính toán trong các hàm khác, còn "hàm hạng nhất" là một thuật ngữ của ngành khoa học máy tính mô tả các thực thể của ngôn ngữ lập trình trong đó không có giới hạn về việc sử dụng (vì vậy các hàm hạng nhất có thể xuất hiện ở bất cứ đâu trong chương trình, giống như các thực thể hạng nhất khác nhau con số, trong đó có cả việc làm tham số cho các hàm khác và làm giá trị trả về của hàm khác).

Các hàm bậc cao cho phép áp dụng bán phần hoặc currying, một kỹ thuật trong đó hàm lần lượt sử dụng từng tham số của nó, mỗi lần sử dụng lại trả về một hàm mới và chấp nhận tham số tiếp theo. Việc làm này cho phép người lập trình biểu diễn một cách súc tích hàm số kế thừa, tương tự như toán tử cộng sẽ lần lượt cộng từng số tự nhiên lại với nhau.

## Hàm thuần túy

Các hàm (hoặc biểu thức) thuần túy hàm không có bộ nhớ hoặc các hiệu ứng lề nhập/xuất. Điều này có nghĩa là các hàm thuần túy có một số đặc tính hữu ích, mà đa số trong chúng có thể dùng để tối ưu mã nguồn:

* Nếu kết quả của một biểu thức thuần túy không được sử dụng, ta có thể xóa nó đi mà không ảnh hưởng đến các biểu thức khác.
* Nếu một hàm thuần túy được gọi cùng với các tham số không tạo ra hiệu ứng lề, kết quả sẽ là hằng số tương ứng với danh sách tham số cụ thể, tức là hàm thuần túy nếu được gọi lần nữa với cùng bộ tham số, kết quả trả về cũng sẽ y hệt như trước.
* Nếu không có sự phụ thuộc về dữ liệu giữa hai biểu thức thuần túy, thì thứ tự của chúng có thể đảo cho nhau, hoặc chúng có thể thực hiện song song mà không ảnh hưởng đến nhau.
* Nếu toàn bộ ngôn ngữ không cho phép hiệu ứng lề, thì chiến thuật đánh giá hàm nào cũng dùng được. Việc này trao cho trình biên dịch quyền tự do sắp xếp lại hoặc phối hợp việc đánh giá biểu thức trong một chương trình.

Trong khi đa số trình biên dịch dành cho các ngôn ngữ lập trình mệnh lệnh có thể nhận dạng hàm thuần túy, và thực hiện các phép khử biểu thức con thường gặp trong các lệnh gọi hàm thuẫn túy, chúng không thể lúc nào cũng làm như vậy đối với các thư viện đã dịch sẵn, thường không tiết lộ thông tin này, vì thế làm cản trở sự tối ưu hóa liên quan đến các hàm bên ngoài như vậy. Một số trình biên dịch, như gcc, thêm từ khóa bổ sung để giúp lập trình viên đánh dấu hàm nào là hàm thuần túy, để cho phép tối ưu hóa kiểu như vậy. Fortran 95 cho phép hàm được chỉ định "thuần túy".

## Đệ quy trong lập trình hàm

Vòng lặp trong các ngôn ngữ hàm thường được thực hiện thông qua đệ quy. Hàm đệ quy sẽ tự gọi chính nó, cho phép thực hiện đi thực hiện lại một tác vụ. Việc đệ quy có thể đòi hỏi phải sử dụng một chồng (stack), nhưng đệ quy đuôi vẫn có thể được trình biên dịch nhận ra và tối ưu hóa nó thành cùng đoạn mã được dùng để hiện thực vòng lặp trong ngôn ngữ mệnh lệnh. Tiêu chuẩn của ngôn ngữ Scheme là phải nhận diện và tối ưu hóa được đệ quy đuôi. Một trong những cách tối ưu hóa đệ quy đuôi là chuyển chương trình thành kiểu truyền liên tiếp trong quá trình dịch.

Các mẫu đệ quy phổ biến đều có thể được khử đệ quy bằng các hàm bậc cao, catamorphism và anamorphism (hay "fold" và "unfold" - gấp và mở gấp) là những ví dụ rõ nhất. Các hàm bậc cao như vậy đóng vai trò tương tự như các cấu trúc điều khiển có sẵn như vòng lặp trong ngôn ngữ mệnh lệnh.

Đa số các ngôn ngữ lập trình hàm đa mục đích đều cho phép đệ quy không giới hạn và là Turing complete, khiến cho bài toán dừng trở nên không quyết định được, có thể gây ra sự thiếu căn cứ cho việc suy diễn công thức, và nói chung đòi hỏi phải có khái niệm không nhất quán trong logic do hệ thống kiểu của ngôn ngữ quy định. Một vài ngôn ngữ lập trình với mục đích đặc biệt như Coq chỉ cho phép đệ quy well-founded và chuẩn hóa mạnh (tính toán không dừng chỉ có thể biểu diễn bằng dòng giá trị vô hạn gọi là codata). Kết quả là, những ngôn ngữ như vậy không phải Turing complete và một số hàm không thể biểu diễn trong ngôn ngữ, dù ngôn ngữ đó vẫn có thể biểu diễn được rất nhiều cách tính toán thú vị mà tránh được vấn đề do đệ quy không giới hạn gây ra. Lập trình hàm giới hạn trong việc đệ quy well-founded với một số ràng buộc khác được gọi là lập trình hàm hoàn toàn.

## Tính toán chặt và không chặt

Có thể chia các ngôn ngữ hàm làm hai loại tùy vào việc chúng sử dụng cách tính toán biểu thức chặt (tham lam) hay không chặt (lười biếng), là những khái niệm chỉ cách xử lý thông số của hàm khi tính toán một biểu thức. Sự khác biệt về các cách tính toán này xuất hiện ở ngữ nghĩa biểu thị của biểu thức khi chúng có chứa phép toán lỗi hoặc có vấn đề. Khi tính toán chặt, việc tính toán số hạng có chứa lỗi cũng sẽ dẫn đến lỗi.

Nói một cách ngắn gọn, tính toán chặt luôn luôn tính toán tất cả cấc số hạng của hàm trước khi xử lý hàm. Tính toán không chặt không tính toán tham số của hàm trừ khi nó cần giá trị đó để tính toán hàm.

Cách hiện thực thông thường của tính toán không chặt trong ngôn ngữ hàm là thu giảm đồ thị. Cách tính toán không chặt được dùng mặc định trong vài ngôn ngữ lập trình hàm thuần túy, như Miranda, Clean và Haskell.

Hughes vào năm 1984 đã phản bác việc dùng tính toán không chặt làm cơ chế để tăng tính module hóa của chương trình thông qua quá trình chia nhỏ bài toán, bằng cách làm cho việc hiện thực độc lập giữa nhà sản xuất và khách hàng dễ dàng hơn. Launchbury 1993 mô tả một số khó khăn mà đánh giá lười biếng tạo ra, cụ thể trong việc phân tích yêu cầu lưu trữ của chương trình, và đề xuất ngữ nghĩa hoạt động để hỗ trợ cho việc phân tích này. Harper 2009 đề xuất đưa cả tính toán chặt lẫn không chặt vào một ngôn ngữ, bằng cách dùng hệ thống kiểu của ngôn ngữ để phân biệt chúng.

## Kiểu, tính đa kiểu, kiểu dữ liệu đại số và so trùng mẫu

Các ngôn ngữ hàm có một hệ thống kiểu dữ liệu hoàn toàn khác với các ngôn ngữ mệnh lệnh. Trong các ngôn ngữ mệnh lệnh (như Pascal, C, Ada..), tính định kiểu tĩnh chặt chẽ (static strong typing) bắt buộc người lập trình phải mô tả kiểu cho mỗi biến dùng đến. Sau khi khai báo, người sử dụng không được thay đổi kiểu dữ liệu của biến trong khi chạy chương trình.

Các ngôn ngữ hàm thường không sử dụng định kiểu. Tuy nhiên một số ngôn ngữ hàm lại sử dụng định kiểu động (dymamic typing) hoặc định kiểu ẩn chặt chẽ (implicit strong typing) (thường gặp trong ngôn ngữ Fortran).

Như vậy, các ngôn ngữ hàm tạo ra được tính mềm dẻo về các hàm đa kiểu (polymorphic functions) với lời gọi có các tham đối có các kiểu dữ liệu khác nhau.

Tính đa kiểu là yếu tố rất quan trọng trong nhiều ngôn ngữ hàm. Với mỗi tham biến hình thức (formal parameters), một hàm đa kiểu có thể chấp nhận lời gọi tương ứng với nhiều tham đối thực sự (actual parameters) có các kiểu khác nhau. Khác với các thủ tục trong Ada, một hàm đa kiểu là một hàm đơn (single function), không phải là các hàm bội (multiple functions) có cùng tên

Đặc biệt kể từ sự phát triển của luận kiểu Hindley–Milner trong thập niên 1970, các ngôn ngữ lập trình hàm có khuynh hướng sử dụng phép tính lambda định kiểu, chống lại phép tính lambda bất định kiểu đã được dùng trong Lisp và các biến thể của nó (như Scheme). Việc sử dụng các kiểu dữ liệu đại số và so trùng mẫu làm cho việc thao tác các cấu trúc dữ liệu phức tạp trở nên thuận tiện và rõ ràng hơn; sự tồn tại của việc kiểm tra kiểu mạnh mẽ trong thời gian biên dịch làm cho các chương trình trở nên đáng tin cậy hơn, trong khi đó luận kiểu giải phóng lập trình viên khỏi việc cần phải khai báo thủ công các kiểu để biên dịch.

Một số ngôn ngữ lập trình hàm định hướng nghiên cứu như Coq, Agda, Cayenne, và Epigram dựa trên lý thuyết kiểu intuitionistic, thuyết này cho phép các kiểu phụ thuộc vào các term. Các kiểu như vậy được gọi là các kiểu phụ thuộc. Các hệ thống kiểu này không có các luận kiểu khả định và rất khó để hiểu và lập trình với chúng. Nhưng các kiểu phụ thuộc này có thể mô tả các mệnh đề tự do trong logic mệnh đề. Thông qua Curry–Howard isomorphism, sau đó, các chương trình định kiểu tốt trong những ngôn ngữ này sẽ trở thành các phương tiện cho việc viết các chứng minh toán học hình thức mà từ đó một trình biên dịch có thể sinh ra mã được chứng nhận. Trong khi những ngôn ngữ này chủ yếu được quan tâm trong nghiên cứu học thuật (kể cả trong toán học hình thức hóa), chúng cũng bắt đầu được sử dụng trong kĩ thuật. Compcert là một trình biên dịch cho một tập con của ngôn ngữ lập trình C được viết bằng Coq và đã được chứng thực chính thức.

Một dạng giới hạn của các kiểu phụ thuộc được gọi là kiểu dữ liệu đại số được khái quát hóa (GADT). Dạng này có thể được thực hiện theo cách cung cấp một vài trong số các lợi ích của lập trình phụ thuộc kiểu trong khi tránh hầu hết sự bất tiện của nó. GADT có sẵn trong Trình biên dịch Glasgow Haskell và trong Scala (như "case classes"), và được cho là phần bổ sung vào các ngôn ngữ khác bao gồm cả Java và C#.

## Lập trình hàm trong ngôn ngữ phi hàm

Có thể sử dụng phong cách hàm của việc lập trình trong các ngôn ngữ mà theo truyền thống không được xem là ngôn ngữ hàm. Một số ngôn ngữ phi hàm đã mượn nhiều đặc điểm như các hàm bậc cao hơn, và các quan niệm danh sách từ các ngôn ngữ lập trình hàm. Điều này làm cho việc chấp nhận phong cách hàm dễ dàng hơn khi sử dụng những ngôn ngữ này. Các cấu trúc hàm như các "hàm bậc cao hơn" và các "danh sách lười" có thể lấy được trong C++ qua các thư viện. Trong C, các con trỏ hàm có thể được dùng để đạt được một vài trong số các hiệu quả của các hàm bậc cao hơn. Ví dụ hàm chung map có thể được thực thi bằng cách dùng các con trỏ hàm. Trong Visual Basic 9 và C# 3.0 và cao hơn, các hàm lambda có thể được dùng để viết các chương trình theo phong cách hàm. Trong Java, các lớp nặc danh đôi khi được sử dụng để mô phỏng các sự đóng, tuy nhiên các lớp nặc danh không phải luôn luôn là các thay thế chính xác cho các sự đóng bởi vì chúng có nhiều khả năng bị hạn chế hơn.

Nhiều mẫu thiết kế hướng đối tượng có thể biểu đạt bằng các thuật ngữ lập trình hàm: ví dụ, mẫu chiến lược đơn giản chỉ ra cách dùng của hàm bậc cao hơn, và mẫu khách viếng thăm gần như tương ứng với một catamorphism, hoặc fold.

Các ích lợi của dữ liệu bất biến có thể được thấy ngay cả trong các chương trình mệnh lệnh, vì thế các lập trình viên thường xuyên cố gắng làm cho một số dữ liệu bất biến ngay cả trong các chương trình mệnh lệnh.

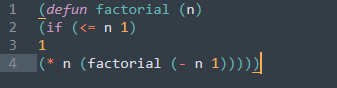
# GIỚI THIỆU VỀ NGÔN NGỮ LISP

Cho đến nay, các ngôn ngữ hàm đã được phát triển là Lisp, ISWIM, FP, Scheme, Common Lisp, Hope, Standard ML, Miranda, Haskell và Lucid.

Ngôn ngữ Lisp (List processing languages) được đề xuất bởi Mc Carthy từ năm 1958. Rất nhiều khái niệm về lập trình hàm được phát triển từ ngôn ngữ Lisp. Lisp hiện đại lại không phải là ngôn ngữ hàm thuần túy, mà cho phép sử dụng các phép gán cho biến, và do vậy, có thể xảy ra hiệu ứng phụ.

Tuy nhiên, Lisp là ngôn ngữ đầu tiên mang phong cách lập trình hàm. Lisp cho phép viết các hàm đệ quy và các hàm bậc cao. Danh sách là cấu trúc dữ liệu chủ yếu của Lisp. Việc cấp phát và giải tỏa bộ nhớ được tiến hành tự động, nhờ kỹ thuật dọn rác (garbage collection).

Ngôn ngữ Lisp chủ yếu được phát triển cho các ứng dụng ký hiệu (symbolic applications), đặc biệt trong lĩnh vực trí tuệ nhân tạo (artificial intelligence). Đối với các phép tính số thì Lisp sử dụng không hiệu quả vì tốc độ tính toán chậm. Một điểm yếu khác của Lisp là các chương trình Lisp thường rất khó đọc, khó hiểu, do cú pháp sử dụng các dấu ngoặc là chủ yếu. Ví dụ, hàm factorial tính giai thừa, được viết trong chương trình Lisp như sau :



Cú pháp của Lisp rất đơn điệu (uniform), tuy nhiên, điều này giúp cho việc lập trình và lưu giữ các chương trình Lisp trở nên dễ dàng như là dữ liệu chuẩn. Yếu tố này cũng giúp cho người lập trình dễ viết các trình sửa lỗi (debugger) và những công cụ tiện ích ngay trong Lisp.

Một vấn đề khác đối với ngôn ngữ Lisp là sự thiếu vắng hệ thống khai báo kiểu. Một danh sách có thể chứa nhiều kiểu phần tử khác nhau. Điều thoải mái này có thể có lợi trong một số trường hợp, nhưng lại gây khó khăn cho trình biên dịch khi cần phát hiện các lỗi sai về sử dụng kiểu dữ liệu. Lisp thường sử dụng kỹ thuật động để cấp phát bộ nhớ vật lý cho biến (dynamic binding) nên rất dễ gây ra sự lộn xộn, nhầm lẫn khi sử dụng biến.

Ngôn ngữ Lisp là tổ tiên trực tiếp của nhiều ngôn ngữ khác, có nhiều ảnh hưởng đến cách thiết kế các ngôn ngữ hàm. Các ngôn ngữ thừa kế Lisp sử dụng hệ thống kiểu, cho phép sử dụng các kỹ thuật tính giá trị hàm theo kiểu khôn ngoan và so khớp.

# TÀI LIỆU THAM KHẢO

1. Lập trình hàm, < <https://vi.wikipedia.org/wiki/L%E1%BA%ADp_tr%C3%ACnh_h%C3%A0m> >
2. Trở thành Functional Programmer phần 1,2,3 < https://viblo.asia/p/tro-thanh-functional-programmer-phan-1-PqOzvNbWzAB >

# PHÂN CÔNG & ĐÁNH GIÁ CÔNG VIỆC THÀNH VIÊN NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Thành viên** | **Nhiệm vụ** | **Mức độ hoàn thành** |
| 1 | Dương Văn Sơn | Tìm hiểu cơ sở lý thuyết về lập trình hàm | Hoàn thành |
| 2 | Nguyễn Thanh Tuân |
| 3 | Trần Nhật Trường | Tìm hiểu ngôn ngữ LISP |
| 4 | Nguyễn Vũ Thành Tuyển |