INTRODUCTION

Sau khi hoàn thành môn học này, học sinh có thể:

mô tả chính thức từ vựng và ngữ pháp của một

ngôn ngữ lập trình

mô tả và giải thích một số cơ chế của một ngôn ngữ lập trình

triển khai một trình thông dịch / trình biên dịch cho một ngôn ngữ lập trình

Tăng khả năng diễn đạt ý tưởng

Cải thiện nền để lựa chọn thích hợp ngôn ngữ

Tăng khả năng học ngôn ngữ mới

Hiểu rõ hơn về tầm quan trọng của thực hiện

Sử dụng tốt hơn các ngôn ngữ đã biết

Tiến bộ tổng thể của máy tính

Ứng dụng khoa học

Fortran, ALGOL 60

Ứng dụng kinh doanh

СОBOL

Trí tuệ nhân tạo

LISP, Prolog

Lập trình hệ thống

PL / S, BLISS, ALGOL mở rộng và C

Phần mềm web

XHTML, JavaScript, PHP

Đặc điểm ngôn ngữ Language Characteristics

Sự đơn giảnSimplicity

Trực giaoOrthogonality

Hỗ trợ trừu tượng (Điều khiển, Dữ liệu)*Support of abstraction (Control, Data)*

Sự an toànSafety

...

*Đánh giá (evaluation)*

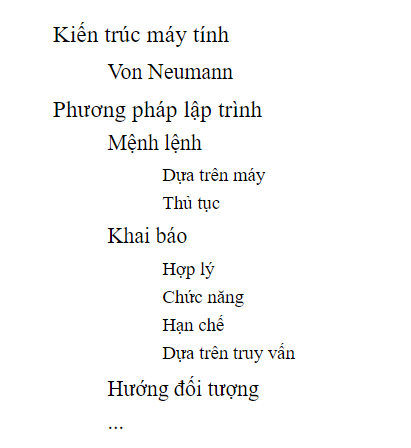
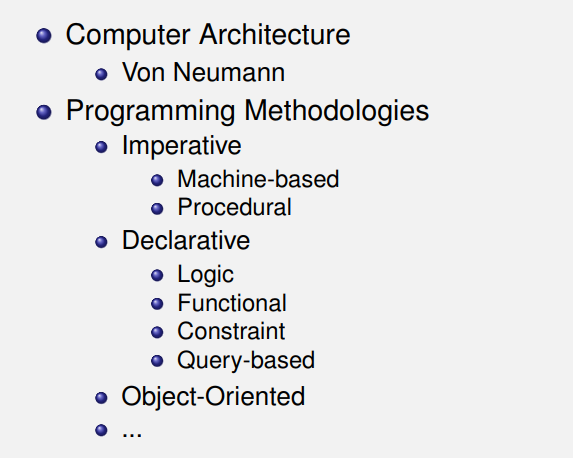
Khả năng đọcReadability

Writability

độ tin cậyReliability: phát hiện được lỗi sai

Phí tổnCost : chi phí để học 1 NNLT, phí tổn để lập trình, phát triển sản phẩm, chi phí bảo trì

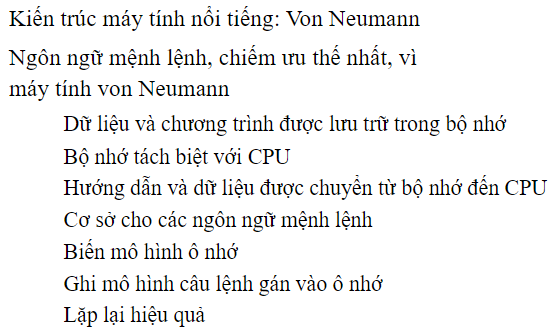
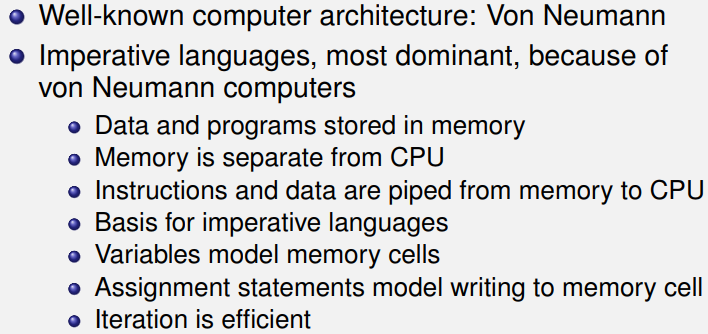
Ảnh hưởng đến thiết kế ngôn ngữ (Influences on Language Design)



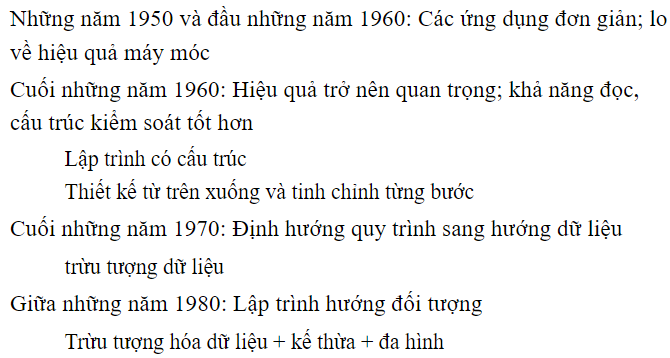
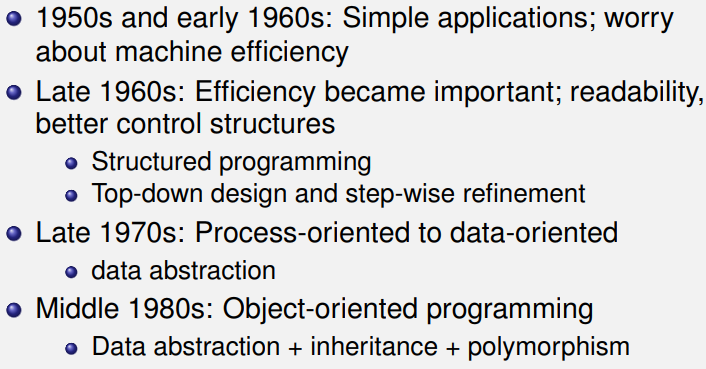
Lập trình thủ tục: trả lời câu hỏi how, làm sao để giải quyết vấn đề

Lập trình khai báo: trả lời câu hỏi what, vấn đề là gì (VD: SQL)

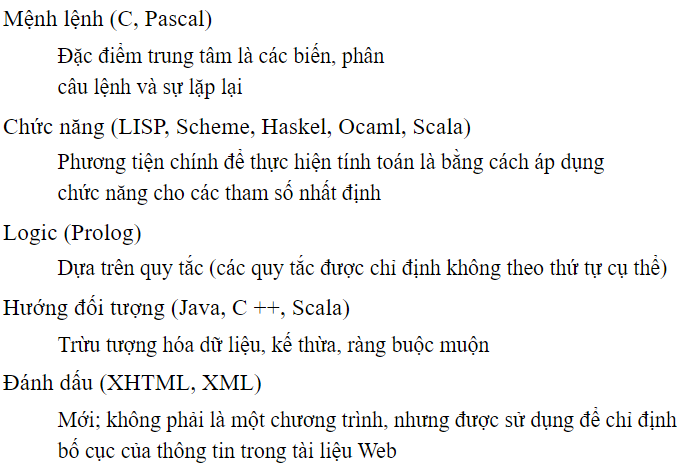
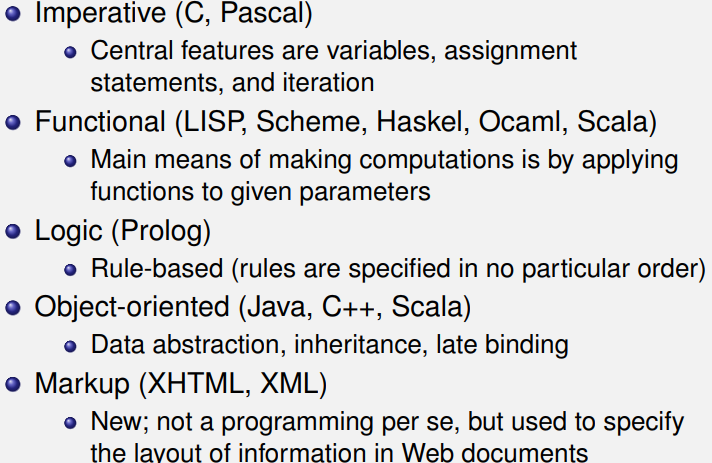
Ảnh hưởng của kiến trúc máy tính: Computer Architecture Influence



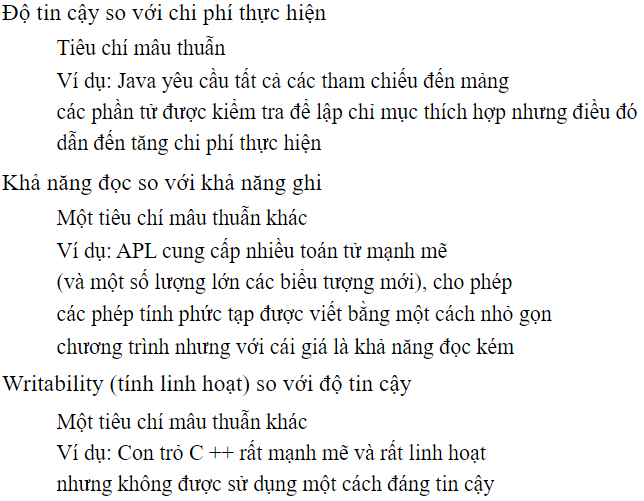
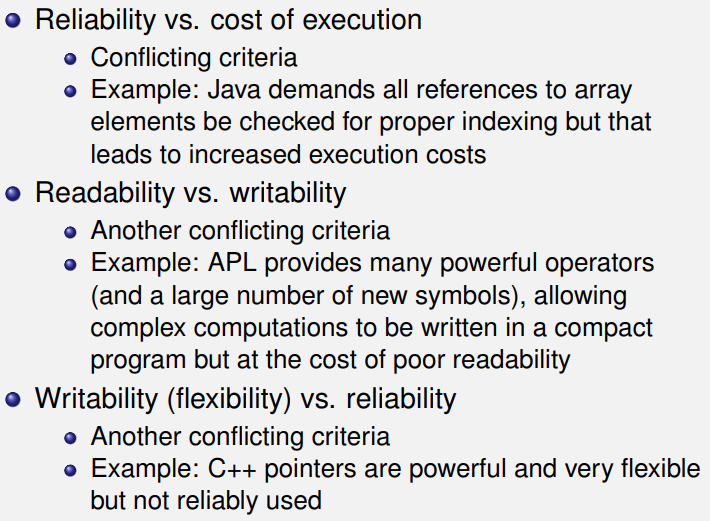
Phương pháp lập trình Ảnh hưởng: Programming Methodologies Influences



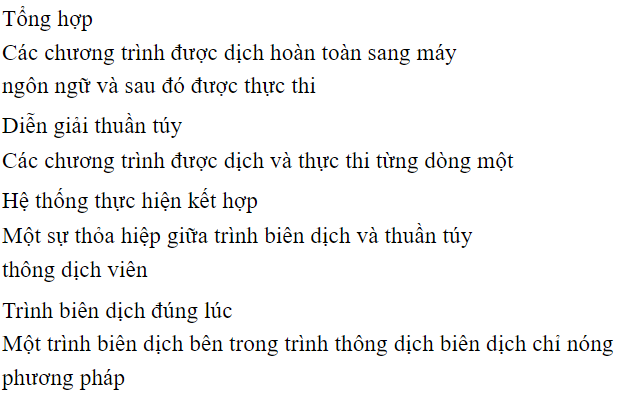
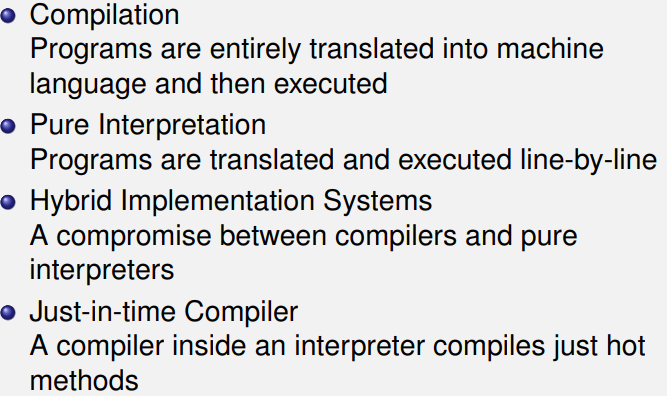
Mô hình ngôn ngữ: Language Paradigms



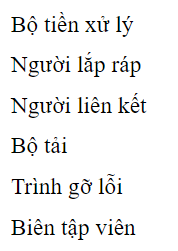
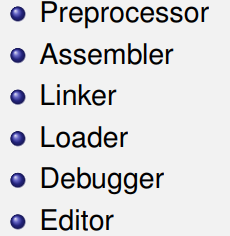
Giao dịch thiết kế ngôn ngữ: Language Design Trade-Offs



Phương pháp thực hiện: Implementation Methods

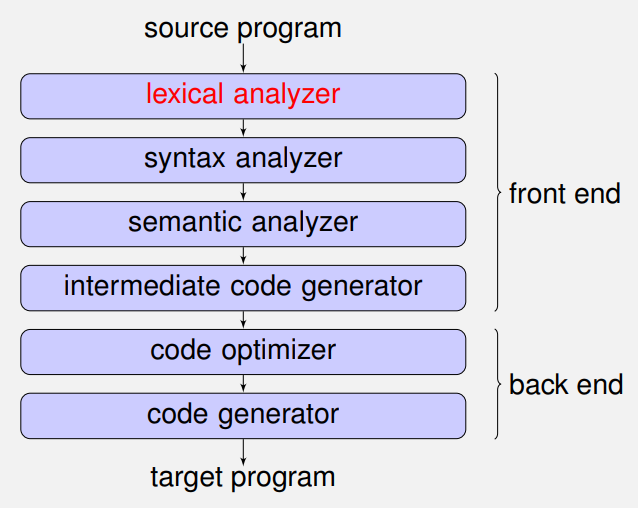


Các chương trình liên quan: Related Programs



Lexical Analysis

Phân tích từ vựng



Phân tích từ vựng

Phân tích cú pháp

Phân tích ngữ nghĩa

Trình tạo mã trung gian

Trình tối ưu mã

Trình tạo mã

Nhiệm vụ:

- giống như 1 bộ tách từ: tìm các từ trong chuỗi liên tục

-giống như bộ kiểm tra chính tả

-giống như 1 bộ phân lớp: phân loại từ

Vai trò của phân tích từ vựng:

-xác định được những chuỗi con thuộc đơn vị ngữ pháp

-trả về token loại từ đó

-bỏ đi những khoảng trắng, dòng mới, tab

-ghi nhận lại vị trí của các token được sử dụng trong lần tiếp theo

Để mô tả các luật về từ vựng, có 2 phương pháp được sử dụng phổ biến là:

- Automata hữu hạn. (cấp máy)

- Biểu thức chính qui. (gần gũi với con người)

Rule Representations(Trình bày quy tắc)

Finite Automata (Dữ liệu tự động hữu hạn)

• Deterministic Finite Automata (Dữ liệu tự động hóa hữu hạn xác định)

• Nondeterministic Finite Automata (Dữ liệu tự động hữu hạn không xác định)

Regular Expressions (Biểu thức chính quy)

Finite Automata gồm:

-1 tập hữu hạn các trạng thái

- 1 trạng thái đọc 1 ký tự chuyển sang trạng thái mới ---> đến khi đến trạng thái kết thúc (đúng với automata hữu hạn)

Automaton hữu hạn xác định (DFA) là một bộ 5

M = (K, Σ, δ, s, F) trong đó

K = một tập hợp hữu hạn của trạng thái ( a finite set of state )

Σ = bảng chữ cái (alphabet)

s €K = 1 trạng thái ban đầu (the initial state)

F *⊆* K = tập hợp các trạng thái cuối cùng (the set of final states)

δ = một hàm chuyển đổi từ K × Σ sang K (a transition function from K *×*Σ to K )

Automaton hữu hạn không xác định (Nondeterministic Finite Automata)

Cho phép một số "trạng thái tiếp theo" có thể có cho một sự kết hợp của trạng thái hiện tại và ký hiệu đầu vào

Chấp nhận chuỗi trống € trong biểu đồ trạng thái: đi từ trạng thái này sang trạng thái kia k cần đọc ký hiệu nào khác

Giúp đơn giản hóa mô tả về dữ liệu tự động

Mỗi NFA tương đương với một DFA

Regular Expression (regex)

Mô tả các tập hợp chuỗi thông thường

Các ký hiệu khác ngoài () | \* đại diện cho chính họ

Sử dụng e cho một chuỗi trống

Nối a B = Phần thứ nhất khớp với a, phần thứ hai phần B

Liên minh a | B = So khớp a hoặc B

Ngôi sao Kleene a \* = 0 hoặc nhiều hơn các kết quả trùng khớp của a

Sử dụng () để nhóm

a + = một hoặc nhiều (tức là aa \*)

a? = 0 hoặc 1 (tức là (a | €))

[xyz] = x |y|z

[xy] = tất cả các ký tự từ x đến y, ví dụ [0-9] = tất cả ASCII

chữ số

[^ xy] = tất cả các ký tự ngoài [xy]

. phù hợp với bất kỳ ký tự nào

ANTLR: trình tạo phân tích cú pháp / lexer mạnh mẽ

SYNTAX ANALYSIS

các cấp độ khác nhau mà tại đó có thể xuất hiện mô tả của một ngôn ngữ: Grammar, semantics, pragmatics

Ngữ pháp là phần mô tả ngôn ngữ trả lời các câu hỏi: cụm từ nào đúng?

Ngữ nghĩa là một phần của mô tả ngôn ngữ tìm cách trả lời câu hỏi "một cụm từ chính xác có nghĩa là gì?"

Ngữ dụng là một phần của ngôn ngữ de- tập lệnh tự hỏi "làm thế nào để chúng ta sử dụng một câu có nghĩa?"

Trong trường hợp ngôn ngữ lập trình, chúng ta có thể thêm cấp độ thứ tư vào cấp độ ba

những cái cổ điển: mức độ thực hiện . Cho rằng các ngôn ngữ mà chúng tôi quan tâm

là các ngôn ngữ thủ tục (nghĩa là các ngôn ngữ có các cụm từ chính xác chỉ định các hành động),

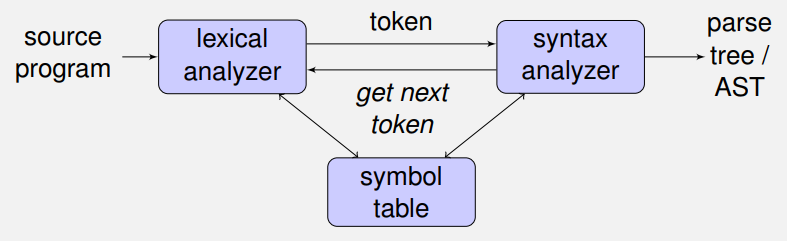
nó vẫn còn để chúng tôi mô tả "làm thế nào để thực hiện một câu đúng, theo cách mà

chúng tôi tôn trọng ngữ nghĩa ".

Syntax analyer nằm sau lexical analyzer

Syntax analyzer đóng vai trò chủ đạo, khi nào cần token thì sẽ gửi yêu cầu “get token” cho lexical analyzer

Mỗi lần chạy lexical analyzer chỉ trả về 1 token, lý do khi nào syntax analyzer cần thì sẽ gọi lexical analyzer, lexical analyzer sẽ đọc chương trình nguồn cho đến khi tìm thấy 1 token, nó sẽ trả token về cho syntax analyzer. Syntax analyzer nhận được sẽ tiếp tục phân tích, khi nào cần token mới sẽ yêu cầu. Trong quá trình thực hiện sẽ tạo nên bảng symbol table.



Vai trò của syntax analyzer:

-Đọc chuỗi token

-nếu chuỗi đúng, sẽ tạo ra 1 cây phân tích cú pháp (a parse tree) hoặc 1 cây cú pháp trừu tượng (abstract syntax tree AST)

-nếu chuỗi sai, tạo ra Error messages.

-Xác định trật tự của chuỗi tokens có phù hợp với qui định của ngôn ngữ không

Biểu thức chính quy không thể mô tả loại cấu trúc này. Cần:

Một phương tiện để mô tả loại ngôn ngữ này

Một phương pháp để phát hiện xem một chuỗi mã thông báo có hợp lệ hay không

không hợp lệ liên quan đến loại ngôn ngữ này

Phương pháp mô tả mạnh hơn biểu thức chính quy: văn phạm phi ngữ cảnh (A context-free grammar (CFG)) là công cụ giúp mô tả được những ngôn ngữ có cấu trúc đối xứng, đệ quy bao gồm bộ 4:

- tập hợp kết thúc T: là những token mà bộ phân tích từ vựng đã trải qua

-tập hợp không kết thúc N: những khái niệm văn phạm đưa ra dùng để mô tả cấu trúc của 1 câu

-Ký hiệu bắt đầu S∈N

-tập các luật sinh P có dạng: X→ a

trong đó X €N và a là một chuỗi các ký hiệu trong T và / hoặc N

-1 văn phạm dùng để mô tả 1 ngôn ngữ

-dựa vào văn phạm người ta có thể nhận dạng 1 chuỗi nhập có thuộc 1 ngôn ngữ hay k

-từ 1 văn phạm làm sao biết được ngôn ngữ đó là ngôn ngữ gì, những chuỗi ngôn ngữ đó là những chuỗi nào, bài toán thứ 1 là bài toán sinh, bài toán dẫn xuất

-Quá trình nhận dạng: cho 1 chuỗi, chuỗi đó có được sinh bởi văn phạm đó hay k?

Làm thế nào để biết ngôn ngữ mà CFG mô tả?

Bắt đầu với chuỗi chỉ chứa ký hiệu bắt đầu

Thay thế bất kỳ ký hiệu không phải đầu cuối X nào trong chuỗi bằng

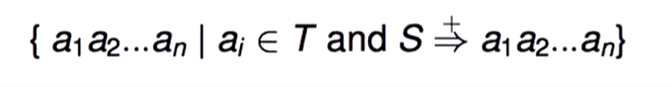
bên phải của một số sản xuất X –→ a

Lặp lại (2) cho đến khi không có thiết bị đầu cuối nào trong chuỗi

Note: => ký hiệu thay thế

Tập hợp các chuỗi có thể được tạo ra bởi 1 cách thức nào đó từ văn phạm phi ngữ cảnh được gọi là ngôn ngữ phi ngữ cảnh

Gọi G là ngữ pháp không có ngữ cảnh với ký hiệu bắt đầu S. ngôn ngữ L (G) được tạo ra từ G là:



1 vài mẹo:

Dựa trên đặc điểm ngôn ngữ

Cố gắng tìm ra cấu trúc phân cấp của một ngôn ngữ

Tập trung vào thứ tự của các đơn vị cú pháp thay vì

ý nghĩa hoặc các ràng buộc khác

Sử dụng đệ quy để mô tả một cái gì đó xảy ra nhiều

lần

Sử dụng đệ quy để mô tả các cấu trúc lồng nhau

Dựa trên đặc điểm ngôn ngữ

Cố gắng tìm ra cấu trúc phân cấp của một ngôn ngữ

Tập trung vào thứ tự của các đơn vị cú pháp thay vì

ý nghĩa hoặc các ràng buộc khác

CFG chỉ giúp mô tả thứ tự của các mã thông báo

CFG không thể được sử dụng để mô tả các ràng buộc kiểu

Một số loại ràng buộc khác như phạm vi, tên

độ phân giải, ... không thể được giải quyết bởi CFG

Sử dụng đệ quy để mô tả một cái gì đó xảy ra nhiều

Lần

Sử dụng đệ quy để mô tả <many> dựa trên <one>

<many> → <one> [<separator>] <many>

| <trường hợp không đệ quy>

<separation>: biểu tượng được sử dụng để phân tách các phần tử trong danh sách

a , c10 , d: integer

hàm foo (a: integer; b: real)

<trường hợp không đệ quy>: số phần tử tối thiểu trong

danh sách

<one> nếu có ít nhất một phần tử hoặc có một

<khoảng cách> giữa các phần tử trong danh sách

e nếu danh sách có thể trống và không có <phân tích>

giữa các phần tử trong danh sách

Sử dụng đệ quy để mô tả các cấu trúc lồng nhau

Trong ngôn ngữ lập trình, có nhiều cấu trúc lồng nhau:

Có thể có một khai báo hàm trong một hàm

tờ khai

Một tuyên bố có thể xuất hiện bên trong một lệnh khác.

Một khối có thể xuất hiện bên trong một khối khác

Một biểu thức có thể là một toán hạng của một biểu thức khác

Ví dụ: sử dụng đệ quy để mô tả cấu trúc lồng nhau

<stmt> → IF <exp> THEN <stmt> ELSE <stmt>

| WHILE <exp> LÀM <stmt>

…

Biểu mẫu Backus-Naur mở rộng (EBNF)

cho phép sử dụng toán tử trong biểu thức chính quy trong RHS (vế phải)

biểu cảm cao hơn

thường được hồ trợ bởi trình tạo phân tích cú pháp từ trên xuống

Chương: OOP

Mục tiêu lập trình hướng đối tượng

Các chương trình dưới dạng tập hợp các đối tượng cộng tác

Đối tượng có giao diện công khai, triển khai ẩn

Các đối tượng được phân loại theo hành vi của chúng

Các đối tượng có thể đại diện cho các thực thế hoặc thực thể trong thế giới thực sản xuất dịch vụ cho một chương trình

Các đối tượng có thể được sử dụng lại trên các chương trình

Giới thiệu về ngôn ngữ OOP

Có rất nhiều ngôn ngữ lập trình hướng đôi tượng (OOP)

Một số là ngôn ngữ OOP thuần túy (ví dụ: Smalltalk).

Các ngôn ngữ mới hơn không hỗ trợ các mô hình khác nhưng

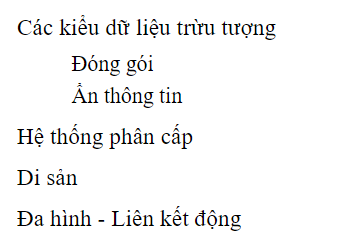
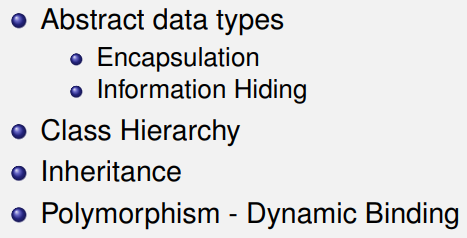
sử dụng cấu trúc mệnh lệnh của chúng (ví dụ: Java và C#).

Một số hỗ trợ thủ tục và hướng dữ liệu

lập trình (ví dụ: Ada và C ++).

Một số chương trình chức năng hỗ trợ (ví dụ: Scala)

Một số tính năng quan trọng của OOP



Kiểu dữ liệu trừu tượng là kiểu dữ liệu đáp ứng sau hai điều kiện:

Đóng gói:

Các thuộc tính và hoạt động được kết hợp trong một đơn vị cú pháp

biên dịch riêng

Ấn thông tin:

Quyền truy cập vào các thành viên được kiểm soát

Tăng độ tin cậy

Phân cấp lớp:

Một lớp có thể có một số Lớp con (subclasses)

Một lớp có thể có một / nhiều lớp cha (supperclasses)

Kế thừa:

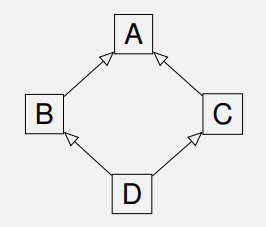
Một lớp con có thế kế thừa các thành viên không riêng tư của lớp cha của nó

Một lớp con có thể thêm các thành viên của chính nó và ghi đè các thành viên kế thừa của nó

Đơn kế thừa so với đa kế thừa

Tính kế thừa làm tăng khả năng tái sử dụng trong OOP

Vấn đề kim cương trong thừa kế nhiều người:



nếu D kế thừa từ B và C các phiên bản khác nhau của

cùng một hành vi, phiên bản nào sẽ hiệu quả trong D?

vấn đề này, được gọi là vấn đề kim cương, đã được giải quyết

khác nhau bằng ngôn ngữ OO khác nhau

có vấn đề kim cương trong Java không?

Class vs Object

Một lớp xác định các đặc điểm trừu tượng của một thứ

thuộc tính hoặc thuộc tính của nó và

các hành vi hoặc phương pháp hoặc tính năng của nó.

Chó có bộ lông và có thể sủa

Một đối tượng là một thể hiện cụ thể của một lớp.

Lassie là một con chó

Method vs Message

Một phương thức mô tả một hành vi của một đối tượng.

Một con chó có thể sủa

Thông báo là một quá trình mà tại đó một phương thức của một

đối tượng được gọi.

Lassie sủa

Đa hình:

Tính đa hình: các đối tượng khác nhau có thể phản hồi cùng một thông điệp theo những cách khác nhau.

Ràng buộc động

Có hai loại biến trong một lớp:

Biến lớp

Biến thể hiện

Có hai loại phương thức trong một lớp:

Phương thức lớp - chấp nhận thông báo cho lớp

Phương thức phiên bản - chấp nhận thông báo tới các đối tượng

Bài 5: FP

-Hàm được xem là các giá trị, 1 hàm có thể là:

+ẩn danh:

Bài 7: NAME, BINDING AND SCOPE

1. Định nghĩa:

1. -name: chuỗi ký tự được sử dụng để đại diện cho 1 cái gì đó khác

+định danh, toán tử (+,&,\*)

-sử dụng biểu tượng thay vì địa chỉ để chỉ 1 thực thể

-trừu tượng

1. Binding (ràng buộc): hoạt động liên kết 2 thứ

Binding time(thời gian ràng buộc): thời điểm mà việc ràng buộc được thực hiện

\*một số vấn đề:

+ràng buộc sớm so với ràng buộc muộn: xảy ra từ khi thiết kế ngôn ngữ $ đưa ct vào thực thi thì mới xảy ra

+ràng buộc tĩnh so với ràng buộc động: xãy ra trước khi ct thực thi (biến cần tên gì, kiểu gì) $ thời gian chạy

+ràng buộc đa hình: 1 tên liên kết với nhiều hơn 1 thực thể

+bí danh: nhiều tên được liên kết với 1 thực thể

\*binding time:

+thời gian thiết kế ngôn ngữ (người thiết kế ngôn ngữ đưa ra ràng buộc)

+thời gian thực hiện ngôn ngữ (khi viết ct dịch sẽ quyết định [còn thiếu trong thiết kế]) (kiểu nguyên 2 byte 4 byte)thời gian triển khai ngôn ngữ là khoảng thời gian mà trình biên dịch hoặc trình thông dịch của ngôn ngữ được xây dựng

+thời gian lập trình (do người lập trình quyết định) x kiểu nguyên, tên biến,…)

+thời gian biên dịch (khi dịch sẽ thực hiện ràng buộc đó) giữa lệnh gọi và ct con được gọi

+thời gian liên kết (khi viết ct trên nhiều file khác nhau, khi dịch sẽ tạo ra nhìu file object, sau đó sẽ có ct Link editer nối các file object lại với nhau)

+thời gian tải (nạp ct từ đĩa cứng) địa chỉ tương đối (link) địa chỉ tuyệt đối (load)

 thời gian nạp là lúc loader nạp chương trình thực thi từ bộ nhớ ngoài vào bộ nhớ trong. Tuỳ theo thực tế của bộ nhớ trong lúc chương trình được nạp mà chương trình sẽ được cấp phát vào một vùng nhớ phù hợp. Sau khi cấp phát thì các biến toàn cục mới có địa chỉ tuyệt đối và khi đó, ràng buộc mới xảy ra,

+thời gian chạy (lúc chạy mới xảy ra) giá trị 1 biến được nhập vào khi ct chạy [một chương trình khi thực thi sẽ có 3 vùng nhớ: vùng nhớ tĩnh, stack và heap. Các biến toàn cục được cấp phát trong vùng nhớ tĩnh nên địa chỉ tuyệt đối của chúng sẽ được xác định trước khi thực thi.]

\*object lifetime(thời gian tồn tại của đối tượng)

-đối tượng: bất kỳ thực thể nào trong chương trình

-thời gian tồn tại của đối tượng: khoảng thười gian giữa quá trình tạo ra và phá hủy đối tượng

-thời gian ràng buộc

-tham chiếu dangling (đối tượng dữ liệu thì mất nhưng tên vẫn còn)

P=new int;

Q=p;

Delete p;

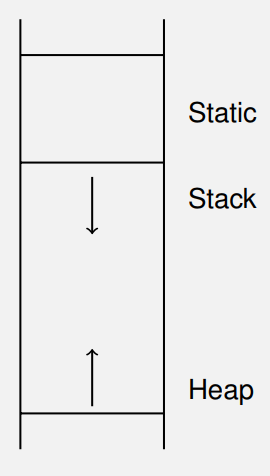
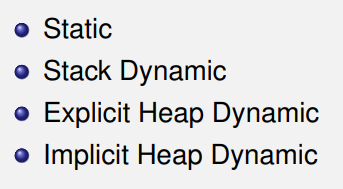
\*q;

-rò rỉ bộ nhớ-garbage  (đối tượng dữ liệu không thể truy xuất)

P=new int;

P=null;

\*Object allocation:



-static: đối tượng được cấp phát trong vùng nhớ cố định, tính toán khi được dịch. Thời gian sống kéo dài trong suốt thời gian thực thi ct (biến toàn cục, hằng, chuỗi)

-stack dynamic: (vùng chung giữa stack và heap) cấp phát theo quy luật cái nào tạo trước hủy bỏ sau, quản lý dễ (đỉnh stack) phù hợp với quy luật tạo trc hủy sau

-explicit heap dynamic: 1 lệnh cụ thể tạo ra đối tượng (tường minh)

-implicit heap dynamic: k có lệnh cụ thể, tự thân ct ngầm định tạo ra đối tượng trên vùng heap

=>heap: k theo quy luật nào, tạo hủy tùy chọn

\*khối: 1 khối là vùng văn bản, có thể chứa các khai báo cho vùng đó

\*phạm vi: của 1 ràng buộc là vùng văn bản của chương trình mà ràng buộc có hiệu lực

-tĩnh so với động:

+phạm vi tĩnh, hoặc phạm vi từ vựng được xác định trong quá trình biên dịch

=ràng buộc hiện tại: trong khối bao quanh chặt chẽ nhất

=phạm vi toàn cục

=phạm vi tĩnh cục bộ

+phạm vi động: được xác định trong thời gian chạy:

=ràng buộc hiện tại: lần thực thi gần đây nhất nhưng không bị phá hủy

\*quy tắc phạm vi static cho các khối

-tham chiếu đến 1 số nhận dạng luôn bị ràng buộc với khai báo cục bộ nhất của nó

-1 khai báo mà ẩn bên ngoài khối mà nó xuất hiện

-các khai báo trong các khối bao quanh có thể nhìn thấy trong các khối bên trong, trừ khi chúng đã được khai báo lại

-các khối có thể được đặt tên và khai báo tên của nó được coi là 1 khai báo cục bộ của khối bên ngoài

\*môi trường tham chiếu:

-môi trường tham chiếu cảu 1 câu lệnh là tập hợp tất cả các tên hiển thị cho câu lệnh

-trong ngôn ngữ có phạm vi tĩnh, đó là các tên cục bộ cộng với tất cả các tên hiển thị trong tất cả các phạm vi bao quanh

-trong 1 một ngữ có phạm vi động , môi trường tham chiếu là các liên kết cục bộ cộng với tất cả các liên kết hiển thị trong tất cả các chương trình con đang hoạt động

Chương 6: ABSTRACT SYNTAX TREE

\*định nghĩa:

- biểu diễn dạng cây của cấu trúc cú pháp trừu tượng của mã nguồn

- khác với cây cú pháp cụ thể (parse tree) bởi 1 số chi tiết bị bỏ qua

- giúp các giai đoạn phụ không phụ thuộc vào quá trình phân tích cú pháp (parse)

