プログラミング言語論

プログラミング言語の特徴と分類

水野嘉明

1

目次

2

1. プログラミングパラダイム
（programming paradigm）

3

1. プログラミングパラダイム

- *プログラミングパラダイム*
  - プログラミング言語の考え方を根本的に規定している概念的枠組み
  - どのようにプログラムを記述するか、その記述方法の概念

4

1. プログラミングパラダイム

- プログラミングパラダイムのいろいろ
  - 命令型言語（手続き型言語）
  - 関数型言語
  - 論理型言語
  - オブジェクト指向言語

5

1. プログラミングパラダイム

- アスペクト指向
- 契約プログラミング
- ジェネリックプログラミング
- イベント駆動型プログラミング
- 並列プログラミング
- 文芸プログラミング

etc.

6

1. プログラミングパラダイム

- 注意:
  - 他にも、いろいろある
  - 各パラダイムは、必ずしも排他的なものではない

  ⇒ 1つの言語が、複数のパラダイムに対応していることもある

7

2. 命令型言語
（imperative language）

8

2.1 命令型言語の理論モデル

- 最初の高級言語 Fortran の目的は
  - アセンブラ言語による開発の労力を削減すること

- 機械語の影響
  - 四則演算、代入、手続き

9

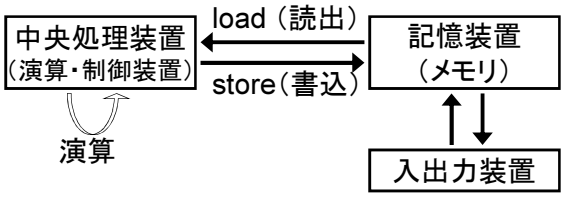
2.1 命令型言語の理論モデル

- 命令型言語実行時の計算モデル
  ＝ ノイマン型コンピュータ
  - 基本的な命令
    - メモリの読み書き（load, store）
    - 演算（add, sub, mul, div など）
    - 制御命令（call, jump など）

10

2.1 命令型言語の理論モデル

- ノイマン型コンピュータのハードウェア構成



11

2.1 命令型言語の理論モデル

- 基本的な動作は、メモリからデータを読出し、演算を行い、結果をメモリに書き込むことである

```
load    A
add     B
store   C
```

これを一般化・抽象化すると

```
C := A + B;
```

代入文 となる

12

2.1 命令型言語の理論モデル

- *命令型言語* は
  - 命令（つまり代入文）の繰り返しにより、変数の値（「状態」という）を動的に変化させ、計算を行う

13

2.2 主な命令型言語

- 主な命令型の言語
  - Fortran → PL/I
  - Algol → Pascal → Modula2
  - Algol → CPL → BCPL → C → C++
  - その他多数

14

2.3 手続き型言語としての側面

- 命令型言語は、別名 *手続き型言語*（procedual language）ともいう
  - *手続き*（procedure）とは
    - 実行すべき一連の計算ステップを持つもの
  - 機械に対する操作（手続き）の記述を連ねたものが、プログラム

15

2.3 手続き型言語としての側面

- 一般的に、手続きは
  - プロシージャ、サブルーチン、メソッド、関数　などと呼ばれる
- 大規模なプログラミングにおいては、手続きによる*モジュール化*が重要
  - 問題の分割 ／ コードの再利用

16

3. 関数型言語
（functional language）

17

3.1 関数型言語

- 関数を定義し、それらの関数を組み合わせることにより問題を表現する
- 引数に対して関数を適用（apply）することにより計算を行う
  - *λ計算*を理論的基盤とする

18

3.1 関数型言語

- 関数型言語は、関数を数値と同じくデータとして扱うことができる
  - 引数や戻り値とすることもできる
- LISPが 最初の関数型言語
  - 1958～1959 J.McCarthy
- その後の、Scheme、ML、Haskell 等も、関数型言語

3.1 関数型言語

- *高階関数* (higher order function)
  - 引数や戻り値が関数である関数
- *参照の透明性* （透過性）
  (referential transparency)
  - 関数の値は引数だけに依存し、いつ評価しても値は変わらない、という性質

20

3.1 関数型言語

- 注
  - 関数型言語でいう「関数」は、命令型言語の「関数」とは、異なる
    - 手続きの一種というより、数学的な関数に近い
    - Cは、典型的な命令型言語であり、関数型言語ではない

21

3.1 関数型言語

- 関数型言語には、次の2種類がある
  - 純粋関数型言語
    - 変数や状態という概念が存在しない
    - Haskell など
  - 非純粋関数型言語
    - 変数を用いることができる
    - ML、LISP など

22

3.2 λ記法

- *λ記法* とは
  - 仮引数となる変数を明示した、関数の表記法
    - 関数をデータとして扱うための道具立て

3.2 λ記法

- 演算記号は、演算を行う関数とみなす

例1: x + y ⇒ +(x, y) ⇒ + x y

例2: sq(x + y) * sin(a * π)

⇒ * sq + x y sin * a π

24

3.2 λ記法

- 関数をλ記法で表す

  sq(x) = x * x

  ⇩

  sq = λx.* x x

λ記法で書いた式を*λ式*という

引数がxであることを示す

関数値の計算方法＝関数本体（body）

25

3.2 λ記法

- 関数適用と関数抽象

  M、Nがλ式のとき

  - MN を *関数適用*（application)という
    - 関数の呼出しに相当する
  - λx.M を *関数抽象*（abstraction)、（または ラムダ抽象）という
    - 関数の定義に相当する

26

3.2 λ記法

- 関数の適用

  sq(3) は

  (λx.* x x)3　と書く

  ⇩

  (λx.* x x)3 = * 3 3 = 9

引数 x を3で置換える

27

3.2 λ記法

- 2引数の関数

  av(x, y) = (x + y)/ 2 は

  ↓

  av = λx. λy./ + x y 2

  ※この式は次のようにみなす

  λx.(λy./ + x y 2)

28

3.2 λ記法

- 関数の適用は 左結合
  - av 3 5 は (av 3) 5
  - av 3 5

    = (av 3) 5

    = ((λx. λy./ + x y 2) 3) 5

    = (λy./ + 3 y 2) 5

    = / + 3 5 2

    = / 8 2

    = 4

29

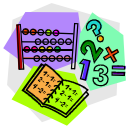
3.2 λ記法

- λ式の定義（M,Nはλ式とする）
  - 変数はλ式である
  - 関数適用(MN) は λ式である
  - 関数抽象(λx.M) はλ式である

※ 紛らわしくない場合は、括弧は適宜省略できるものとする

30

3.2 λ記法



- 演習3.1
  - 次の関数をλ式で表せ
    ただし、演算子は前置記法で書くものとする

(1) f(x) = 2 * x + 1

(2) g(x, y) = a * x * y

31

3.3 λ計算

- λ計算とは
  - 関数の定義と実行を抽象化した計算体系
  - λ式の「簡約」により、計算を行う

★λ計算については、後日詳しくやる

32

4. 論理型言語
（logic programming language）

33

4. 論理型言語

- 公理(論理式)の集合を定義し、推論規則の適用により計算を行う
  - *単一化*(unification)が基本操作
- Prolog が代表
  - 1972 カルメラウア、コワルスキー
  - (実用的な論理型言語は、現在Prologのみ)

34

4. 論理型言語

- Prologの応用
  - 自然言語処理
  - アルゴリズムの記述
  - データベースの探索
  - コンパイラの記述
  - エキスパートシステムの構築
  - ※ パターン照合、後戻り検索、不完全な情報を扱う応用に向いている

35

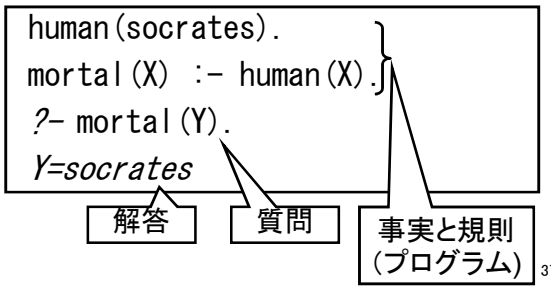
4.1 論理プログラミング

- 論理プログラミングとは
  - (1) 情報の表現のための事実と規則の使用
  - (2) 質問の応答への論理推論の使用
  - プログラマが事実と規則を記述し、質問を与える。処理系が論理推論を用いて質問への解答を計算する

36

4.1 論理プログラミング

- Prologの例



37

4.1 論理プログラミング

- 1行目 human(socrates).
  - 「socrates は human(人間)である」という事実を述べている

注:小文字は定数、大文字は変数

38

4.1 論理プログラミング

- 2行目 mortal(X) :- human(X).
  - 「Xがhumanならば、Xはmortal(死ぬもの)である」という規則
  - 「A :- B.」は,「Bが真ならばAも真である」という*含意*を示す
    - 通常の論理式でいう「B→A」を表している
    - 「A∨¬B」とも書ける

39

4.1 論理プログラミング

- 3行目 *?- mortal(Y).*
  - 目標となる論理式（ゴール式）
  - 「mortalに当てはまるのは誰か？」
- 4行目 *Y=socrates*
  - 実行結果

40

4.1 論理プログラミング

- Prologの規則の記述は、

  $C :- C_1, C_2, \cdots, C_n .$

  - 「C_1～C_nのすべてが成り立てば、Cが成り立つ」という意味
  - 論理式では

    $C \vee \neg C1 \vee \neg C2 \vee \cdots \vee \neg Cn$

    これは、*ホーン節*と呼ばれる

41

4.1 論理プログラミング

- 「 :- 」の左側の述語を*ヘッド部*という
- 右側の述語列を、*ボディ部*と呼ぶ
- １行目の事実の記述は、ヘッド部のみの規則である

42

4.1 論理プログラミング

- 処理系は、与えられた論理式と等価な（同じ値を返す）論理式に変換し、簡素化していく

  これを*導出*（resolution）と呼ぶ

  - *単一化* が、導出の基本操作である

43

4.1 論理プログラミング

- 「単一化」とは
  - *二つの述語を、同じ形にすること*（述語名から引数の値まですべて同じにする）
  - 例
    - f(X, b) と f(a, Y) は単一化可能であり、その結果は f(a, b) である　{X=a, Y=bとした}

44

4.1 論理プログラミング

- 前記3行目⇒4行目の例
  - 質問「mortal(Y)」と 規則のヘッド部「mortal(X)」を単一化　{X=Y}
  - {X=Y} を規則のボディ部にも適用

    human(Y)
  - 「human(Y)」と 1行目「human(socrates)」 を単一化
  - 結果が Y=socrates

45

5. オブジェクト指向言語
（object-oriented language）

46

5. オブジェクト指向言語

- オブジェクト（対象）を定義し、オブジェクト間のメッセージ通信を基本操作とする
- Simula (1964) に始まる
  - その後、Smalltalk、Object Pascal、C++、Java、C#、Ruby などに発展

47

5. オブジェクト指向言語

- オブジェクト指向言語は、次の2種類に分類できる
  - ハイブリッド型
  - 純粋なオブジェクト指向

48

5. オブジェクト指向言語

- ハイブリッド型
  - 命令型言語などに、オブジェクト指向の考え方を追加
  - C++、Object Pascal など
    - Pascal ＋ オブジェクト指向
    - C ＋ オブジェクト指向

49

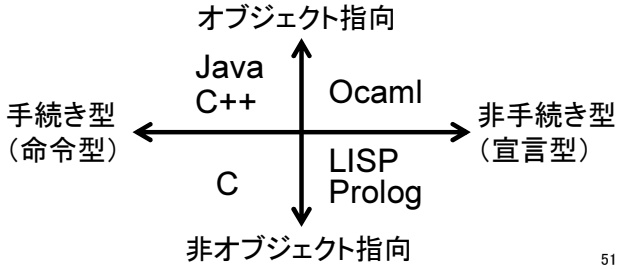
5. オブジェクト指向言語

- 純粋なオブジェクト指向
  - はじめからオブジェクト指向言語として設計されたもの
  - Smalltalk、Java、Ruby など

50

5. オブジェクト指向言語

- 手続き型とオブジェクト指向



51

5.1 オブジェクト指向の考え方

- オブジェクト指向の概念は
  - 高度に抽象化されており、強力なモデル化の能力がある
  - 大規模なシステム開発に向いている
  - プログラミングだけでなく，業務分析や要求定義などの上流工程もカバーする

52

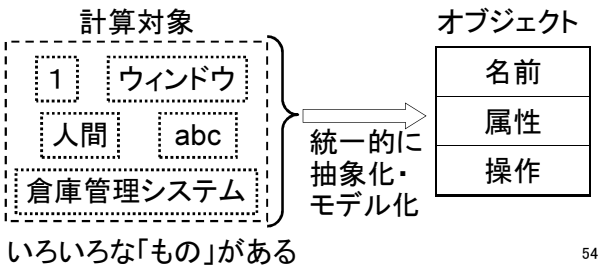
5.1 オブジェクト指向の考え方

- *オブジェクト*（object)とは
  - 実際の「もの」を指す概念
  - プログラム上の計算対象を、抽象化・モデル化したもの
  - 自分自身の名前、属性（データ）、操作（メソッド）を持つもの

53

5.1 オブジェクト指向の考え方

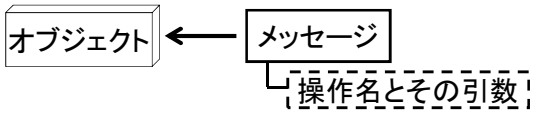
- オブジェクトの概念



54

5.1 オブジェクト指向の考え方

- オブジェクトは、*メッセージ*を送受信する
  - 受信メッセージに従い、操作を行う
  - 操作には、属性の変更や、他のオブジェクトへのメッセージ送信がある



55

5.1 オブジェクト指向の考え方

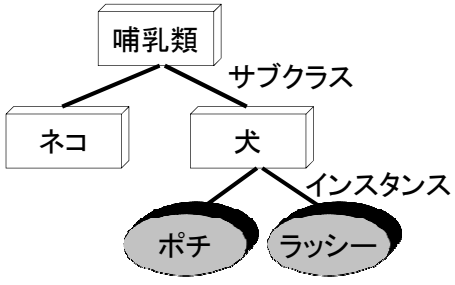
- *クラス*（class）とは
  - オブジェクトを作成するためのテンプレート（雛形）
  - クラスから作製されたオブジェクトを、*インスタンス*（instance）という

※ 型と変数、あるいは概念と個体の関係

56

5.1 オブジェクト指向の考え方

- クラスとインスタンスの例



57

5.1 オブジェクト指向の考え方

- オブジェクト指向プログラミングは大雑把に言うと、
  - クラスを定義し
  - インスタンスオブジェクトを生成
  - 生成したオブジェクトにメッセージを送る

  という流れになる

58

5.1 オブジェクト指向の考え方

- 演習3.2
  - オブジェクト指向におけるクラスとインスタンスとの関係のうち，適切なものはどれか

59

5.1 オブジェクト指向の考え方

ア．インスタンスはクラスの仕様を定義したものである
イ．クラスの定義に基づいてインスタンスが生成される
ウ．一つのインスタンスに対して，複数のクラスが対応する
エ．一つのクラスに対して，インスタンスはただ一つ存在する

（基本情報技術者試験 平22秋 午前 問47）60

5.2 オブジェクト指向言語の特徴

- オブジェクト指向言語の特徴
  - *カプセル化*（*encapsulation*）
  - *継承*（*inheritance*）
  - *ポリモーフィズム*（*polymorphism*
    多態性、多相性）

61

5.2 オブジェクト指向言語の特徴

- カプセル化
  - データ（属性）と操作を、一体のものとして扱う
  - データや操作を*隠蔽*（hiding）する（外部からは直接データを扱うことはできないようにして、安全性を高める）

62

5.2 オブジェクト指向言語の特徴

- 継承
  - あるクラス（子クラス）が別のクラス（親クラス）を基にして作られ、
  - 子クラスが 親クラスの属性や操作を引き継いでいること
    - 子クラスでは、親クラスとの差分だけをプログラミングすればよい

63

5.2 オブジェクト指向言語の特徴

- ポリモーフィズム
  - 一つの操作名で、複数の異なったデータ型（クラス）に対して同様の操作を行うこと

64

5.2 オブジェクト指向言語の特徴

- 例1）リストのサイズは
  list.size()
  で求まり、ベクタのサイズも
  vector.size()
  で求まる
  （どちらのクラスにも、size メソッドが定義される）

65

5.2 オブジェクト指向言語の特徴

- 例2）演算子＋は、数値の和にも文字列の結合にも適用可

1 + 3　　(= 4)

"AB" + "CDE" (= "ABCDE")

66

6. その他のパラダイム

67

6. その他のパラダイム

- アスペクト指向
  - オブジェクト指向の問題点を補うため考えられた
  - クラス間を横断するような機能を、「アスペクト」として分離しモジュール化
  - AcpectC++、AspectJ

68

6. その他のパラダイム

- 契約プログラミング
  - 設計の安全性を高める技法
  - コード中に、プログラムが満たすべき仕様について記述する
    - 事前条件、事後条件、不変条件
  - Eiffel、D言語

69

6. その他のパラダイム

- ジェネリックプログラミング
  - データ形式に依存しないプログラミング方式
  - データ型でコードをインスタンス化する（C++のtemplate）、あるいはデータ型をパラメータとして渡す
  - C++、D、Eiffel、Java など

70

6. その他のパラダイム

- イベント駆動型プログラミング
  - イベントを待ち、発生したイベントに従って処理を行う

    ⇔フロー駆動型プログラミング
  - GUIを使用するプログラムにて、マウス操作やキーボード操作に対する処理に利用

71

お疲れさまでした