プログラミング言語論

第2回
状態モデルと
命令型言語(1)

担当：犬塚

今日の講義

状態モデルと命令型言語パラダイムの特徴を概観する

- 状態モデル
- 命令型言語パラダイム
- 代入文、変数の概念と状態
- 制御文
- 構造化プログラミング
- データ型
- 副プログラムと手続き

状態モデル

- 計算モデル（computation model）

  ＝「計算するとはどういうことか」に対する答え。

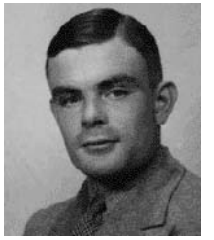
- 状態モデルにおける計算とは
  機械の内部状態を次々に変えてゆくこと。

  要するに、オートマトンを考えること。

チューリング機械

- チューリング機械(Turing Machine)は状態モデルにおける計算の典型例

1. 入力テープを1つ読む（入力記号）
2. 現在の状態と、入力記号の組合せに応じて
   1. ヘッドを動かす（右、左、動かさない）
   2. ヘッド位置に記号を書く
   3. つぎの状態を決める
3. 1に戻って繰り返す



アラン・チューリング

状態モデルと命令型言語パラダイム

- 状態を次々に変化させることで計算をさせる計算モデルでは、命令型プログラミング言語(imperative programming languages)が発展する。
- チューリング機械は状態遷移表を持つが、表が大きくなったとき表形式は得策でない。
- フローチャートは状態遷移図であり、分かりやすいが書き下すのが困難。
- そのため命令型言語が適していると考えられる。

状態モデル＝命令型言語の特徴

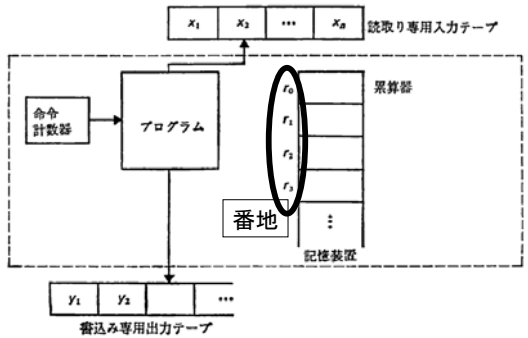
- 変数と変数への代入　→　代入型言語ともいう。
  - 変数と変数の構造
  - データ型
  - データ型と手続きの結びつき→抽象データ型
- 制御構造の発展
  - 構造化プログラミング
- 手続き
  - 手続き呼び出しと値引渡し
  - 再帰呼び出し

変数の概念

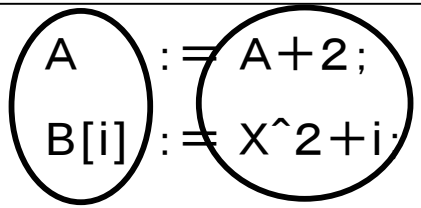
- 変数は記憶装置のある領域に対応する。
- 記憶装置の記憶内容が即ち、その機械の「状態」であるため、状態モデル＝代入(assignment)によるプログラム。
- 記憶装置
  - シーケンシャルアクセスメモリ:決められた順にのみアクセスできる記憶装置 sequential access memory
  - ランダムアクセスメモリ:番地をつかってどの場所も自由にアクセスできる記憶装置 random access memory
  - ノイマン型の計算機はランダムアクセスメモリを主記憶にもつ。

ランダムアクセスマシン（RAM）

- チューリング機械よりも現実に近い計算機のモデル。
- チューリング機械は記憶装置もテープだったのに対し、RAMマシンは番地つきのランダムアクセスメモリをもつ。



代入文



左辺に置かれた式はメモリの場所を示す。
AやB[i]の値に意味があるのでなく、記憶場所を指し示す。
変数が左辺に置かれたとき表す記憶場所を左辺値という。

右辺に置かれた式は値を示す。
A、X、iはメモリそのものでなく、その値を指し示す。
変数が左辺にあるとき表す値を右辺値という。

命令と命令の制御

- 状態モデルでは次の文がある。
  - 状態を変化させる形式、即ち命令文
  - 命令の順序を決める形式、制御構造（制御文）
  - プログラムの解釈を変える形式、宣言文
  - コメント文
- 変数内容の変化が基本であり、他の文はそのための仕掛け。
- 命令文：代入文、入出力文
- 宣言文：変数の宣言、データ型の定義、手続きの定義

命令の制御と制御構造

- 制御構造は、命令の順序を決める文。
- 逐次実行（合成）
  - S1 ； S2
- 分岐
  - if-then、if-then-else、switch-case（多分岐）
- 繰り返し
  - while 条件 do-end (前判定)
  - repeat-until 条件 (後判定)
- （副プログラムの呼出しと復帰）

GOTO文

- GOTO文は無条件分岐。
- 命令にラベルをつけておくとGOTO文は常に、そのラベルの箇所に制御を移す。

```
L1 X:=X+1;
   if X>100 GOTO L2;
   GOTO L1;
L2 …
```

- 機械語にあるため、FORTRAN以来、命令型言語の多くが採用している。
- 制御が自由に変えられるため、多用すると分かりにくいプログラムになる ― スパゲッティプログラム

命令の制御と制御構造

- プログラムは、それぞれの箇所が何をしているのか理解できるように書かなければならない。
  - プログラムのテキストの構造（段落）
  - プログラムの実行順序の構造

  } 一致する必要がある

- 一致するとき、そのプログラムは構造化（structured）されているという。
- 必要な制御構造を容易に記述できる制御文が必要。
  → それを備える言語＝構造的プログラミング言語

structured programming languages

構造的プログラミング

- プログラムのテキスト上の段落と、命令の制御の段落（実行される時間的な固まり）が一致する。

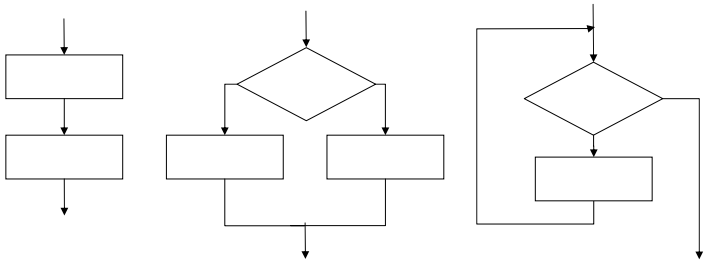
それだけでなく、

- プログラムのテキスト上の段落と、機能的な段落が一致する。
- したがって、読みやすく、保守が容易。(可読性)
- 構造的プログラミング言語といわれる言語で書けばよいわけでない。（心がけの問題）
- GOTO文の排除

構造化された制御フロー

- 逐次実行
- 分岐 if-then、if-then-else
- ループ　while、repeat

の制御形式（あるいはその変形）が代表的。



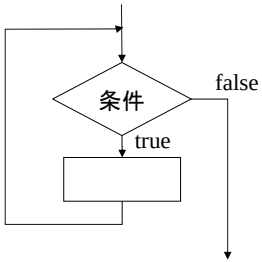
- その段落への入り口、出口が1つのみ。

single entry/single exit

ループの不変条件（不変表明）

- 意味的分かりやすさが構造的プログラミングの利点。
- 特にループ中での性質を明確にすることが重要。
- while文、repeat文ではループ中に満足している条件、脱出したとき満足する条件が明確。

```
while 条件  do
  …
end
```



ループ内では「条件」は必ず真。
＝この条件をループの不変条件 (loop invariant)という。

データ型と変数の型

- データ(値)とデータを入れておく変数は表裏一体。
- これらの種類分けがデータ型(data type)。
- しかし、細かく言えば、
  - 値のデータ型
  - 式のデータ型
  - 変数のデータ型

  がある。
- 値や式にデータ型があっても、変数にはデータ型のない言語もある(Lisp, Prolog)

値の型、式の型、変数の型

PASCALの例:

- 値のデータ型は表現で決まる。
  - 「1」はinteger(整数)、「1. 0」はreal(実数)。
- 式のデータ型は演算子の性質で決まる。
  - 「1＋2. 0」はreal, 「1+2」はinteger。
- 変数のデータ型は、型宣言で決まる。

データ型の働き

- 処理系に情報を与える
  - 必要なメモリの割当て
  - アドレス計算
- エラーの検出
  - データ型の宣言は、変数の利用法を特定するもの。
  - その利用法に合わない場合にエラーを検出する。
  - これを型検査(type check)という。
  - コンパイル時の検査＝静的(static)検査と、実行時(dynamic)の検査がある。
  - 型検査に合格したプログラムを型安全(type safe)という。

データ型:基本型と構造型

- 基本型(basic type):その型のデータを直接扱う演算がある型。
  - 整数型、実数(浮動小数点数)型、文字型、論理型。
- 構造型(structured type):基本型を要素にして作られる型。
  - 配列型、レコード(構造)型、リスト、など。
  - 構造を作るしくみを型構成子(type constructor)という。
  - 言語によっては、集合型、列挙型、関数型などがある。
  - ポインターとそれを用いたデータ構造
- データ型は値の集合に対応、構造型は集合演算に対応する。

副プログラムと手続き

- 命令型言語では次の言葉はおおよそ同じ意味
  - サブルーチン
  - サブプログラム
  - 手続き
  - 関数
- 繰り返し利用するルーチンへ制御を移すものと考えたとき、サブルーチン、サブプログラムが、
- 1つの機能を持つブラックボックスとしてルーチンを抽出したものと考えるとき手続き、関数が好まれる。

手続き、関数

- プログラムの繰り返しを回避するための制御構造ではなく、機能ブロックとしての働きをもたせたルーチンが手続き。
- 手続きが値を返すことを前提とするとき、関数という。

手続き、関数の要素:

- 引数の渡し方、結果の戻し方
- 手続き内部の変数の扱い
- 手続き内外での変数の通用範囲、生存期間

命令型言語パラダイム

- 代入文が基本の命令。
- これの順序を決める制御文を備える。
- 変数やデータの型を有する。
- 副プログラムが書ける。

- 近代的命令型言語では、
- 構造的プログラミングの言語構造をもつ。
- 構造的データ型の型構成子をもつ。
- 手続きに関する言語の取り決めがある。