# iOSプラグラム初級テキスト
# 第３回
# アルゴリズムとデータ構造

Prime Kobo LLC
北島　優

平成26年4月10日

# 目 次

# 第1章　アルゴリズム

## 1.1　アルゴリズムとは

アルゴリズムとは，問題を解くための手順を定式化[1]した形で表現したものです．

問題はその解[2]を持っていますが，アルゴリズムは正しくその解を得る為の具体的手順です．また，同じ解を求めるのにも幾つもの方法が考えられます．その為にアルゴリズムには効率性が重要になります．

[1]手順や法則をある決まった方式に落とし込むこと．定式に当てはめること．
[2]与えられた問題に対する答え．

# 第2章　チューリングマシン

## 2.1　チューリングマシンとは

チューリングマシン [1] とは

1. 無限に長いテープ
2. 上記テープに格納された情報を読み書きするヘッド
3. チューリングマシンの内部状態を記憶しているメモリ

で構成されています.
　内部状態とヘッドから読み出した情報の組み合わせに応じて，以下の動作を行います.

1. ヘッド位置のテープに情報を書き込む
2. チューリングマシンの内部状態を変える
3. ヘッドを右か左に一つ移動する

この動作を，チューリングマシンは内部状態が停止状態になるまで反復して実行し続けます.

## 2.2　万能チューリングマシン

- あるチューリングマシンについて記述したテープを入力とする
- テープの記述に従い，そのチューリングマシンの動作を模倣する

この様なチューリングマシンがあれば，あらゆるチューリングマシンの動作を真似る事ができます．その様なチューリングマシンを万能チューリングマシンと呼びます.（エミュレータの原理です.）

## 2.3　チューリング完全

ある計算のメカニズムが万能チューリングマシンと同じ計算能力をもつとき，その計算モデルはチューリング完全であると言います．

万能チューリングマシンと同等の計算を実現できるという事は，決められた手順に従って計算できるあらゆる問題を処理する事ができる事を意味しています．

一方で，万能チューリングマシンで実現不可能な計算のメカニズムを可能にする方法は知られていません．チューリング完全であれば，実現可能なアルゴリズムや手続きはすべて処理できるという事を意味しています．

多くのコンピュータ言語の内，特にチューリング完全な処理をできる言語の事をプログラミング言語と呼びます．

正規言語やSQLやマークアップ言語等はチューリング完全ではありません．

## 2.4　停止性問題

チューリングマシンは必ず計算を完了できるわけではありません．プログラミングの分野には無限ループと呼ばれ，永遠に処理を続けるものがありますが，計算が止まらない事があります．この様な処理を手続きと呼び，有限の時間内に必ず停止するアルゴリズムと区別します．

# 第3章　アルゴリズムの定式化

アルゴリズムはコンピュータが情報を処理する基盤となります．換言すれば，プログラムは本質的にはアルゴリズムであって，コンピュータが特定の処理（給与計算や成績処理等…何でも良いです）を指定された順序で実行する為のステップをコンピュータに指示します．ですから，アルゴリズムはチューリング完全なシステムで実行可能な操作の並びとみなす事もできます．

情報処理におけるアルゴリズムでは，元となるデータは何らかの入力源（キーボードやマウスやスキャナ等…他にもあります）から読み込まれ，アルゴリズムで得た解は何らかの出力先（モニタやプリンタやデータベース等…）に書かれるか，次の処理の入力となるように保持されます．

保持されたデータはアルゴリズムを実行する実体（プログラム）の内部状態の一部とみなされます．コンピュータでは，この内部状態をデータ構造に保持したりします．

アルゴリズムは，どのような計算過程においても，厳密に定義されなければならず，あり得る全ての状況に適用可能な形で指定しなくてはなりません．即ち，どの様な条件のステップでもケースバイケースで体系的に扱わなければならず，各ケースの扱い方は明確（計算可能）でなければなりません．これは，本来あり得ない異常値が入力された場合にどう振る舞うかをも規定しなければならない事を意味します．

アルゴリズムは明確なステップの明確なリストなので，その計算順序が最も重要になります．命令は先頭から最後尾に向かって逐次的に実行される様に記述されます．この考え方をより形式的にしたものが，先に述べた制御構造です．

## 3.1　停止性

チューリングマシンの考案者である Alan Mathison Turing が停止性問題として提起していますが，任意のアルゴリズムと初期状態が与えられた時に，それが停止するかどうかを判定するアルゴリズム的手続きは存在しません．つまり，そのアルゴリズムが停止するかどうかを判断する手法は無いと言う事になります．

しかし，プログラムにおいては停止してくれなくては困ります．そうでなくては，プログラムは永遠に動き続け，コンピュータは解を返してくれません．気の遠くなりそうな天文学的計算であっても，時間はかかるものの停止性をプログラマが保証しなければなりません．

不完全な（或いは誤った）アルゴリズムは次のいずれかの結果を引き起こします.

- 停止しない
- 解の範囲を逸脱した値を返して停止する
- 誤った解を返して停止する
- 解を返さずに停止する
- これらの組み合わせ

## 3.2　アルゴリズムの表現

アルゴリズムには様々な記法があります．代表的なところでは

自然言語　人間が扱う通常の言語でアルゴリズムを記述します.

疑似コード　特定の言語系に依らず，プログラム言語らしい言語で記述します．場合によっては自然言語が混じる場合があります.

フローチャート　アルゴリズムを特定の記号に則って図示する方法です.

プログラム言語　実際にその時点で開発に用いている言語そのもので記述する.

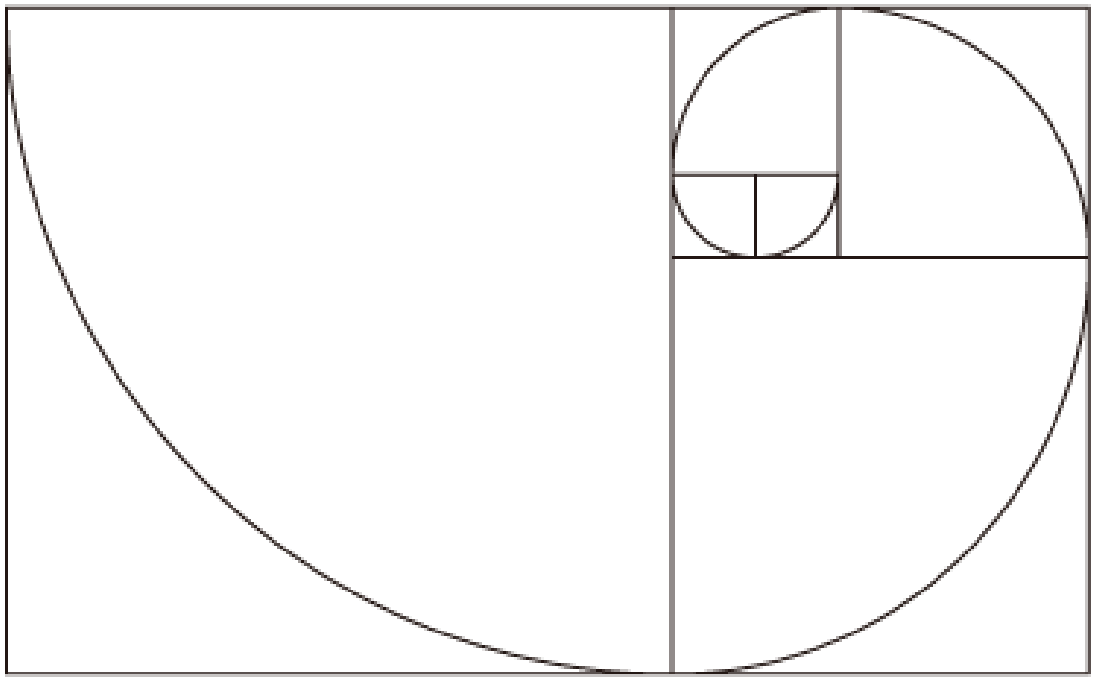
自然言語による表現は冗長で曖昧になる傾向がありますので，複雑なアルゴリズムや技術的な場面では単独ではほとんど使用されません.

擬似コード（Psuedo Code）やフローチャートはアルゴリズムを構造的に表現できて自然言語のような曖昧さが殆どありません.

プログラム言語でアルゴリズムを示す事も比較的多く行われます.

以下にフィボナッチ数列 [2] を例にそれぞれ示します．フィボナッチ数列とは $F_0 = 0, F_1 = 1, F_{n+2} = F_n + F_{n+1}$ で表される数列です．但し，$n \geq 0$ です．

余談になりますが，フィボナッチ数列は不思議で美しいものです．上の定義によれば，フィボナッチ数列は $0, 1, 1, 2, 3, 5, 8, 13, 21 \cdots$ と無限に続きます．第 1 項の 0 を除外して，$F_1$ からを正方形の一辺の長さとして，右回りに結合していくと次の様な図が描けます．



### 3.2.1　自然言語

# 関連図書

[1] Alan Mathison Turing *ON COMPUTABLE THE NUMBERS, WITH AN APPLICATION TO THE ENTSHIEDINGS PROBLEM* (12 Nov. 1936)

[2] Leonardo Fibonacci *Liber Abaci*(1202)