



アルゴリズム入門

目次

1. はじめに	2
1.1. はじめに	2
1.2. 各研修とのつながり	2
2. アルゴリズムとは	3
3. フローチャートとは	4
3.1. フローチャートとは	4
3.2. フローチャートで使用する記号	5
3.3. 変数	6
3.4. 配列	8
3.5. 演算	9
4. フローチャートの記法	11
4.1. 順次構造	11
4.2. 分岐構造	12
4.3. 反復構造	14
4.4. 三大構造の組み合わせ	16

1. はじめに

1.1. はじめに

本資料では、問題を解くための手順である「アルゴリズム」について紹介します。

アルゴリズムを学ぶことで、効率の良いプログラムを作成することができるようになります。

Java や C といった特定の言語だけではなく、どのようなプログラミング言語にも活用できる普遍的な知識になります。プログラミングの基礎と言っても過言ではないので、本資料でしっかりとアルゴリズムについて学びましょう。

1.2. 各研修とのつながり

1.2.1 Java 研修、組込み研修

アルゴリズムを学ぶことで、効率の良いプログラムを作成することが可能になります。言語を問わず、全てのプログラムを作成する際に役に立つ知識になります。

1.2.2 インフラ研修

インフラに関する知識を学ぶため、効率の良いプログラムを作成するための知識であるアルゴリズムは不要に思えますが、そうではありません。

サーバで利用される Unix 系 OS では、シェルスクリプトを利用する場面が多くあります。シェルスクリプトとは、OS を操作するためのプログラミング言語のことです。そのため、シェルスクリプトを作成する際にもアルゴリズムの知識が役に立ってきます。

2. 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」のことです。

コンピュータは単独で動作するものではなく、人間の指示に従って処理を実行します。つまり、コンピュータに何らかの問題を処理させるには「まずはこうして、次にそうして」といった詳細な手順を与える必要があります。

実は、こういった一連の手順はコンピュータだけに限らず、私たち人間の日常にも密接に関わっています。

例えば、料理を作る際、レシピに記述された通りに調理を進めていきますが、レシピがない場合はとんでもない料理が出来上がってしまう可能性があります。コンピュータも同様です。品質の良いプログラムを作るためには、手順（アルゴリズム）が必要不可欠なのです。

また、アルゴリズムというのは、1つだけではありません。同じ料理であったとしても手っ取り早く5分で作る調理法と、手間暇かけて30分で作る調理法が存在するのと同様です。

ここで少し考えてみましょう。朝、出勤前の時間がない時に、皆さんでしたら5分の調理法と30分の調理法、どちらを選択するでしょうか。おそらく後者を選択する方は少ないと思います。

このように、複数存在するアルゴリズムの中から、場面に応じた最適なアルゴリズムを見つけ出すことが何よりも重要となります。

アルゴリズムとは、問題を解く手順のことですが、問題を解くことができればいいというわけではありません。アルゴリズムには、良いアルゴリズムとそうでないアルゴリズムが存在します。以下が良いアルゴリズムの条件になるので、アルゴリズムを考える際の参考にしてください。

(1) 信頼性が高い：精度が良く、正しい結果が得られる

料理が一瞬で出来る調理法でも、美味しくなければ良いアルゴリズムとは呼べません。

(2) 処理効率が良い：計算回数が少なく、処理が速い

美味しい料理でも、非常に時間のかかる調理法は良いアルゴリズムとは呼べません。

3. フローチャートとは

3.1. フローチャートとは

フローチャートとは、「アルゴリズムを図式化したもの」になります。



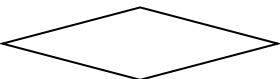

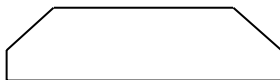



図式化する分手間はかかりますが、以下のようなメリットがあるため、積極的に使用しましょう。

- (1) アルゴリズムを視覚的かつ明確に表すことが出来る
- (2) 処理の手順が追いやすいため問題の発見、修正が容易になる
- (3) 複数人で問題の解決を行う場合、担当の明確化や理解向上に役立つ

3.2. フローチャートで使用する記号

フローチャートは以下の記号を用いて記述します。これらの記号は、日本においては JIS 規格（日本産業規格）で定められています。

表 3-1 フローチャートで使用する記号

記号	名称	説明
	端子	フローチャートの始まりと終わりを表す
	処理	計算、代入などの処理を表す
	判断	1つの入口と複数の出口を持ち、条件によって1つの出口を選ぶ処理を表す。
	線	処理の流れを表す
	ループ始端	反復処理の開始を表す。
	ループ終端	反復処理の終了を表す
	表示	画面表示を表す
	入力	手操作での入力を表す

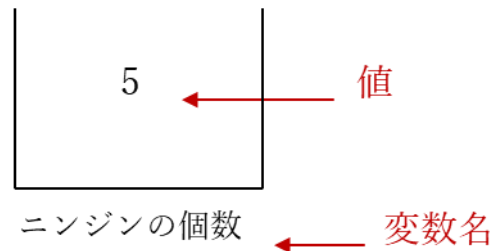
3.3. 変数

先ほど、フローチャートで使用する記号を紹介しましたが、フローチャートを記述するにあたり、理解しておきたいプログラミングの知識があります。ここからは、それらについて説明していきます。

3.3.1 変数とは

変数とは、「データを一時的に記憶するための入れ物」のことです。変数には任意の名前を付けることが出来ます。変数に付けられた名前を「変数名」といい、変数に記憶されているデータを「値」と呼びます。一度変数に記憶された値は、別の値が入れられるまで記憶され続けます。

図 3-1 変数のイメージ

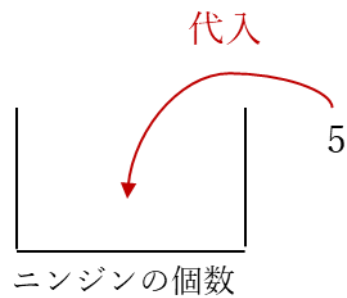


複数のデータを扱う場合や同じデータを何度も利用する場合、計算によってデータを変化させたい場合に変数を利用します。

3.3.2 代入とは

変数は、値を一時的に記憶するための入れ物と説明しました。変数に値を記憶させることを代入といいます。

図 3-2 代入のイメージ



フローチャートで代入を表現すると、以下のような記述になります。

例 1：値「5」を、変数「エンジンの個数」に代入する

「5 → エンジンの個数」

例 2：値「トム」を、変数「シェフの名前」に代入する

「"トム" → シェフの名前」

代入する値が文字列である場合は、値の前後を"（ダブルクォーテーション）で括って下さい。

例 3：変数「ケーキの個数」の値に「1」を加え、その結果を再び変数「ケーキの個数」に代入する

「ケーキの個数 + 1 → ケーキの個数」

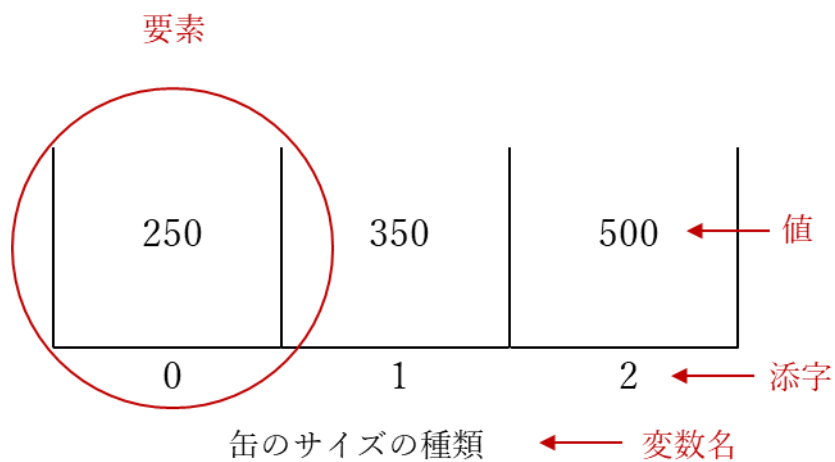
変数には、この例のように変数を代入することも出来ます。

3.4. 配列

変数には 1 つの値しか記憶することが出来ないため、複数の値を扱おうとした場合は複数の変数が必要になり、管理が大変になってしまいます。そのような際に効力を発揮するのが配列です。配列とは「複数のデータを一時的に記憶するための入れ物」で、これを使用することで、複数の値を効率よく管理することが可能になります。

配列は複数の要素から構成され、要素 1 つ 1 つに値を記憶することが出来ます。要素を識別するには添字（インデックス）を用います。たとえば変数「缶のサイズの種類」に値「250」「350」「500」を記憶させた配列は以下のようになります。

図 3-3 配列のイメージ



添字は 0 から始まる点に注意して下さい。例えば、変数「缶のサイズ種類」の「350」は、2 番目ではなく、1 番目ということになります。なお、配列をフローチャートで表現すると、以下のような記述になります。

「250、350、500 → 缶のサイズの種類」

3.5. 演算

「2つの値を計算する」や「2つの値を比較する」といった処理はまとめて「演算」と呼ばれます。

様々な演算を組み合わせることで、複雑な計算結果を求めたり、プログラムに何かを判断させたりすることが可能になります。

プログラム中では、特定の演算を行うために記号を使います。この記号のことを「演算子」と呼びます。演算子には様々な種類がありますが、本資料ではその中でも代表的な「算術演算子」、「比較演算子」を紹介します。

3.5.1 算術演算

「算術演算」とは、「足し算」、「引き算」、「掛け算」、「割り算」などの数の計算を行うことです。特に、基本的な計算である上記4種類は「四則演算」と呼ばれます。

算術演算では、下表のような演算子を使用して計算を行います。

表 3-2 算術演算子

演算子	演算内容	記述例	記述例の演算結果
+	足し算（加算）	5 + 2	7
-	引き算（減算）	5 - 2	3
*	掛け算（乗算）	5 * 2	10
/	割り算（除算）	5 / 2	2
%	割り算の余りを求める（剰余）	5 % 2	1

掛け算と割り算の演算子に注目してください。プログラムの世界では、「×」や「÷」という記号ではなく、「*（アスタリスク）」や「/（スラッシュ）」という記号を使用します。これらの記号は1950年代からプログラムで使われるようになり、今でもその文化が踏襲されています。そのため、私達もプログラムではこれらの記号を使用することになります。

また、算術演算で注意してほしいことがあります。それは、割り算では基本的に「商」しか求められないということです（小数点以下が求められないという場合のみ）。例えば、「5 / 2」という演算では、商である「2」が演算結果となります。

では、割り算の「余り」はどのように求めればよいのでしょうか？割り算の余りは「%（パーセント）」という記号を使用して求められます。例えば、「5 % 2」という演算では、「1」が求められます。

ただし、「%」で求められるのは「余り」のみです。もし、割り算の「商」と「余り」の両方を求めたい場合は、「/」を使用した演算と「%」を使用した演算を個別に行う必要があります。

3.5.2 比較演算

「比較演算」とは、「2つの値が等しいか」や「ある値がもう1つの値より大きいかなど、値の比較結果を判定することです。

比較演算では、下表のような演算子を使用して判定を行います

表 3-3 比較演算子

演算子	演算内容	記述例
==	左の値が右の値と等しいかを判定する	5 == 5
!=	左の値が右の値と等しくないかを判定する	5 != 4
>	左の値が右の値より大きいかを判定する	5 > 3
>=	左の値が右の値より大きいか、または等しいかを判定する	5 >= 3 5 >= 5
<	左の値が右の値より小さいかを判定する	5 < 10
<=	左の値が右の値より小さいか、または等しいかを判定する	5 <= 10 5 <= 5

比較演算を行うと、比較の内容が正しい場合は「真 (True)」、誤っている場合は「偽 (False)」という結果になります。例えば、「5 == 5」という比較は正しいため「真 (True)」、「5 != 5」という比較は誤っているため「偽 (False)」という結果になります。

また、大小関係を比較する際に使用する「>」「<」に「=」がつくと、「その値が等しいか」という条件も含まれるので、注意してください。例えば、「5 > 5」という比較は「偽 (False)」ですが、「5 >= 5」という比較は「真 (True)」になります。

4. フローチャートの記法

アルゴリズムには、3 種類の基本的な構造があります。1 つ目は、順番に処理の流れを示した「順次構造」。2 つ目は、条件によって処理が選択される「分岐構造」。3 つ目は、条件によって同一の処理が繰り返される「反復構造」になります。そして、これら 3 つの構造のことを「3 大構造」と呼びます。どのようなアルゴリズムも、この 3 つの構造の組み合わせで表すことができます。

ここからは、各構造のアルゴリズムをフローチャートで表しています。

4.1. 順次構造

順次構造の例として、A さんから鉛筆を 3 本、B さんから鉛筆を 2 本貰い、貰った鉛筆の合計本数を求める場合について考えてみましょう。この例の場合、以下のようなアルゴリズムになります。

3 → A さんから貰った鉛筆

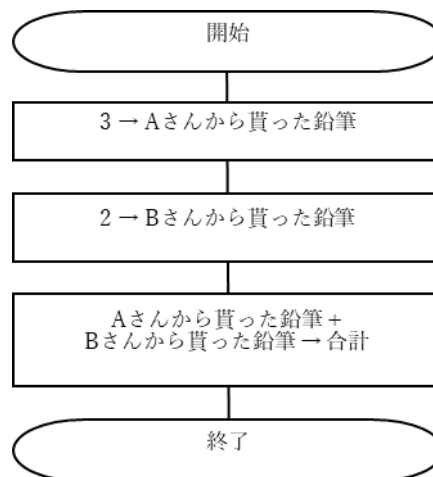
2 → B さんから貰った鉛筆

A さんから貰った鉛筆 + B さんから貰った鉛筆 → 合計

まずは変数「A さんから貰った鉛筆」に 3、次に変数「B さんから貰った鉛筆」に 2 を代入します。そして、それぞれの変数を足し、変数「合計」にその合計値 (5) を代入します。

これをフローチャートで表すと、下図のようになります。まず全体を端子記号で囲い、次に 3 つの処理を処理記号で表し、それを時系列的に線記号で結びます。

図 4-1 順次構造のフローチャート



このように、開始端子記号から終了端子記号まで各処理を直線的に実行することを順次構造といいます。

4.2. 分岐構造

分岐構造の例として、1本100円の鉛筆を3本以上購入したら1割引にするという条件で、鉛筆を5本購入した場合の金額を求める処理について考えてみましょう。

条件により処理が変わる場合には判断記号を用い、処理を分岐させる必要があります。判断記号の中には等号や不等号を用いた条件式を書き、条件式の結果が正しい場合は真(true)、正しくない場合は偽(false)とすることで、処理を分岐させることができます。

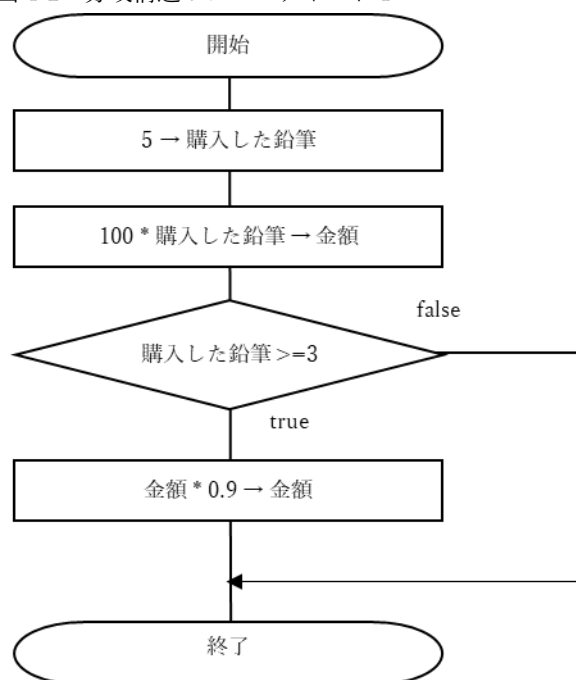
この例の場合、以下のようなアルゴリズム、及びフローチャートになります。

5 → 購入した鉛筆

100 * 購入した鉛筆 → 金額

(もし本数が3本以上ならば) 金額 * 0.9 → 金額

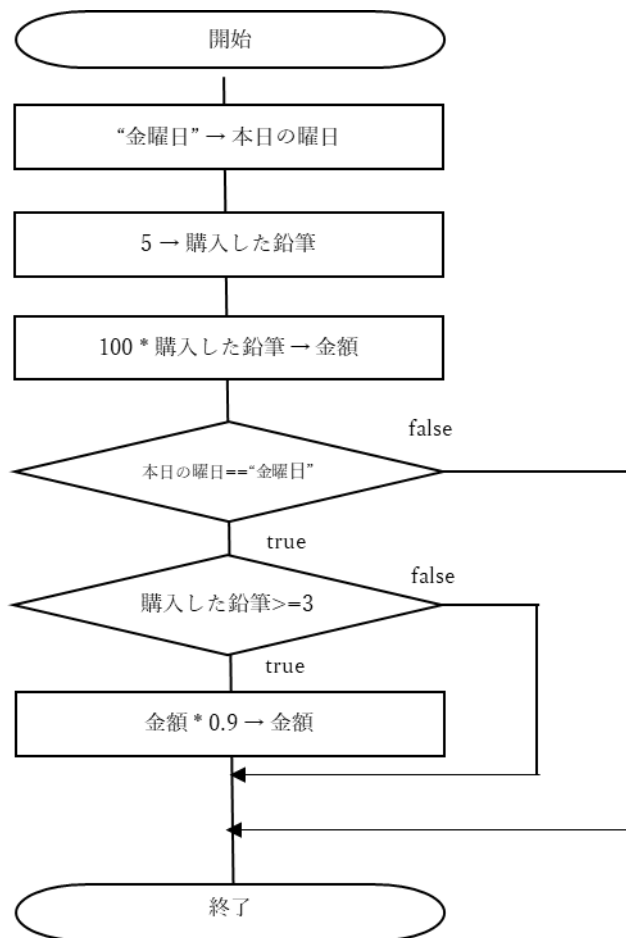
図 4-2 分岐構造のフローチャート 1



判断記号を用いることで処理の流れを変えることができます。このように、場合に応じた処理を行う構造を分岐構造といいます。

なお、判断記号を複数用いることで、より複雑な処理を行うことも出来ます。以下の例を参考に、処理の流れを追ってみましょう。

図 4-3 分岐構造のフローチャート 2



この例は、本日の曜日が金曜日であり、なおかつ鉛筆を 3 本以上購入したら 1 割引にするという条件となります。条件式になっている「本日の曜日 == “金曜日”」は、「本日の曜日が金曜日と等しい」という意味です。

4.3. 反復構造

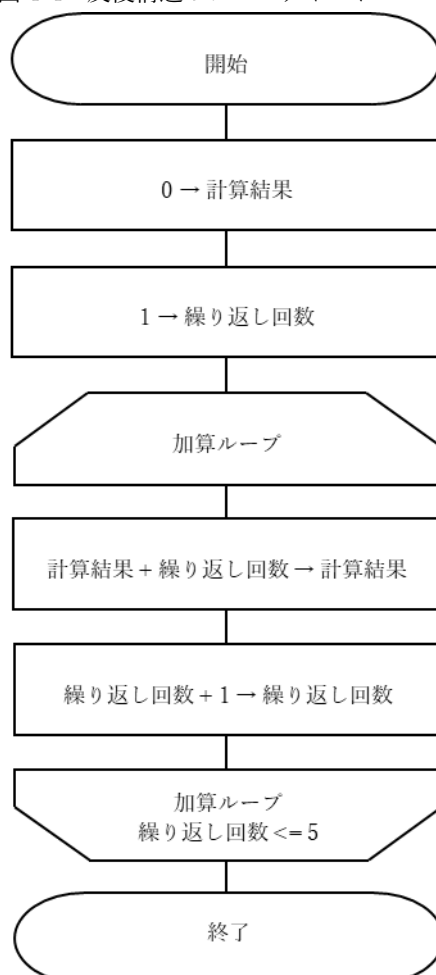
反復構造の例として、 $1 + 2 + 3 + 4 + 5$ の計算結果を求める処理について考えてみましょう。

この処理を簡単に行うのであれば、フローチャートの処理記号の中に、「 $1 + 2 + 3 + 4 + 5 \rightarrow$ 計算結果」と記述すれば OK です。しかし、これが 1 から 100 までの加算の処理になった場合、記述するのに大変な労力を要します。

コンピュータは単純作業が得意ですので、「計算結果に 1 ずつ増やした数を足していく」という方法であれば、機械的に計算を行えます。始めに 1 を足し、次に 2 を足し、更に 3 を足し…といった具合に 5 まで加算を続けます。

このように同じ処理を何度も繰り返す場合には、ループ始端記号及びループ終端記号を用います。ループ始端記号、ループ終端記号には任意の名前を付け、繰り返す処理を間に挟みます。そしてループ始端記号、もしくはループ終端記号の条件式が偽 (false) になるまで処理が繰り返されます。

図 4-4 反復構造のフローチャート



「加算ループ」内の処理についてももう少し詳しく見ていきましょう。ループ内では、以下のような計算が行われています。

0 → 計算結果

1 週目：計算結果(0) + 1 → 計算結果(1)

2 週目：計算結果(1) + 2 → 計算結果(3)

3 週目：計算結果(3) + 3 → 計算結果(6)

4 週目：計算結果(6) + 4 → 計算結果(10)

5 週目：計算結果(10) + 5 → 計算結果(15)

計算式を見てみると、変化しているのは加える数だけになっています。

加える数が5になるまで1ずつ増やし、それを加算することで、「1 + 2 + 3 + 4 + 5」の計算結果を求めています。もし1から100までの加算を求めたいのであれば、ループ終端記号の条件式を「繰り返し回数 ≤ 100」にすればOKです。

なお、ループが終了する条件式を間違えると、永遠に処理を繰り返す無限ループになってしまいます。無限ループになると、強制的に処理を終了しないといけなくなってしまうので、注意が必要です。

また、反復構造には、ループ始端記号に条件式を記述する前判定型ループ、例のようにループ終端記号に条件式を記述する後判定型ループがあります。これらのループの書き方は以下のようになります。

表 4-1 前判定型ループと後判定型ループ

前判定型ループ	後判定型ループ
<p>1 回目から終了条件が成立すると、処理が 1 回も実行されない</p>	<p>少なくとも 1 回は、処理が実行される</p>

4.4. 三大構造の組み合わせ

三大構造を組み合わせることで、複雑なアルゴリズムを表現することができます。

例として、30 歳以上の場合は「30 代です」、20 歳の場合は「20 代です」、20 歳の場合は「未成年です」と出力する処理について考えてみましょう。

この例の場合、以下のようなフローチャートになります。

図 4-5 三大構造を組み合わせたフローチャート

