基本情報技術者

試験対策テキストⅣ【アルゴリズム編】

無料体験入学者用

TAC

はじめに

基本情報技術者試験は，2009年春の情報処理技術者試験の制度改革により，従来のシステム開発技術者を対象とした試験から，システム開発技術者・利用者を問わずITにかかわるすべての人材を対象とした試験に衣替えしました。これに伴い，試験で問われる知識範囲も整理・拡充され，現代のIT社会に必要な幅広い知識を問うようになっています。

本書は，基本情報技術者試験の午後問題の中核となる「データ構造及びアルゴリズム」の内容に対応するテキストです。そして，IT分野について初心者の方でも無理なく学習が行えるよう，基礎的な用語や考え方を分かりやすく解説するように心がけました。

本書により，読者のみなさんが基本情報技術者試験に合格されることを願ってやみません。

2008年10月
TAC情報処理技術者講座

アルゴリズム　目次

Part 1

アルゴリズムの基礎

このPartでは，コンピュータに処理を行わせるための手順である“アルゴリズム”について学習していきます。

1-1 アルゴリズムとは何か

アルゴリズムを直訳すると“算法”となります。一般に「計算の手順」，あるいは「必要な処理結果を得るための手順」を意味します。なお，ここで説明されているすべてを覚えることはありません。概要をつかんでおきましょう。

基本知識の整理

アルゴリズムの意味

アルゴリズムというと，難しい計算をイメージするかもしれませんが，直接的な算術計算だけを意味するものではありません。たとえば，「東京から大阪へ行くための経路を求める」といった問題を解くための手順もアルゴリズムとよんでよいのです。

参考：JISによる定義

JIS(日本工業規格)では，「明確に定義された有限個の規則の集まりであって，有限回適用することにより問題を解くもの」と定義しています。ここで“問題を解く”とは，コンピュータで処理させ，結果を得ることをいいます。より具体的にいえば，「コンピュータに“データ”を入力し，これをプログラムで“処理”することで，利用者に役立つ(より付加価値の高い)“情報”を導くこと」といえます。

複数の計算手順

一般に，ある問題の答えを得るための手順，すなわちアルゴリズムは複数存在します。たとえば，「1から10までの和」を得るには，

1＋2＋3＋…＋9＋10

というように1から順に一つずつ大きい数を足していってもよいですし，逆に

10＋9＋8＋…＋2＋1

というように10から順に一つずつ小さい数を足していってもよいわけです。どちらも答えは55ですね。また，少し工夫すると，

1＋10＝11
2＋ 9＝11
3＋ 8＝11

$4 + 7 = 11$

$5 + 6 = 11$

というように，11の組が五つできますから，

11×5

としても結果は55となります。

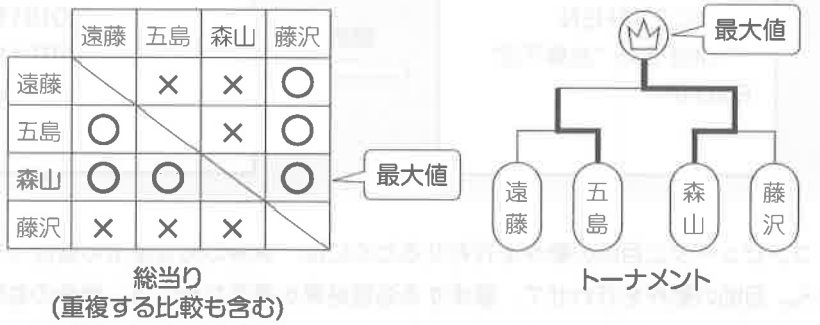
理解を深めよう

アルゴリズムの効率

ある問題を解くためのアルゴリズムは一つとは限りません。例として「受験者の試験結果から最高点を求める」ためのアルゴリズムを考えてみましょう。

最高点は受験者の得点を“総当り”に比較しても求めることができます。総当りの結果，ほかのすべての得点よりも高い得点が最高点となるでしょう。

また，受験者の得点を“トーナメント”で比較することでも最高点を求めることはできます。



しかし，単に最高点を求めるのであれば，総当りで求めるようなことは，まずしません。なぜならば，効率が悪いからです。たとえば4人の受験者がいる場合，総当りに必要な比較回数は，自分自身との比較を除けば，

$4 \times 3 = 12$［回］

となります。つまり，総当りでは4人の受験者の中から最高点を選ぶのに「12回の比較」を行なわなければならないわけです。

これに対して，4人の受験者の得点を“トーナメント”で比較する場合を考えてみましょう。その際の比較回数（試合回数）は，

$2 + 1 = 3$［回］

で済むことになります。

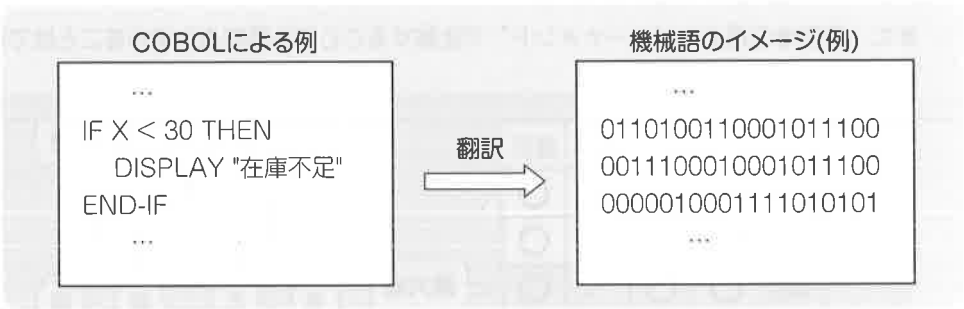
コンピュータのCPUは，こうした比較処理を一つひとつ処理していきますから，同じ結果が得られるのであれば，より少ない処理となるようなアルゴリズムのほうがよいことがわかるでしょう。

発展知識

プログラム言語とアルゴリズム

プログラムは，コンピュータに一連の動作を行わせるための命令の集まりです。コンピュータのCPUは，“0”と“1”の数字の並び（数字列）で表された機械語（マシン語ともいいます）による命令しか理解することができません。“0”と“1”の数字列では，人間が直接理解するには困難を伴います。そこでCやCOBOL，Javaといった，より人間が用いる言語（自然言語という）に近い**プログラム言語**が開発されてきました。このようなプログラム言語のことを**高水準言語**とよびます。

高水準言語は，機械語に比べればはるかに人間が理解しやすいものになっていますが，CPUは高水準言語を直接理解し，その命令を実行することはできません。そこで，**コンパイラ**や**インタプリタ**などの**言語プロセッサ**によって，高水準言語の命令を機械語命令に置き換えてから，実行することになります。



コンピュータに目的の動作を行わせるためには，意味のある命令の並びでなければなりません。目的の動作を行わせて，要求する処理結果を得るためには，命令の並びは処理手順を表すようになっていなければならないのです。この処理手順がアルゴリズムです。

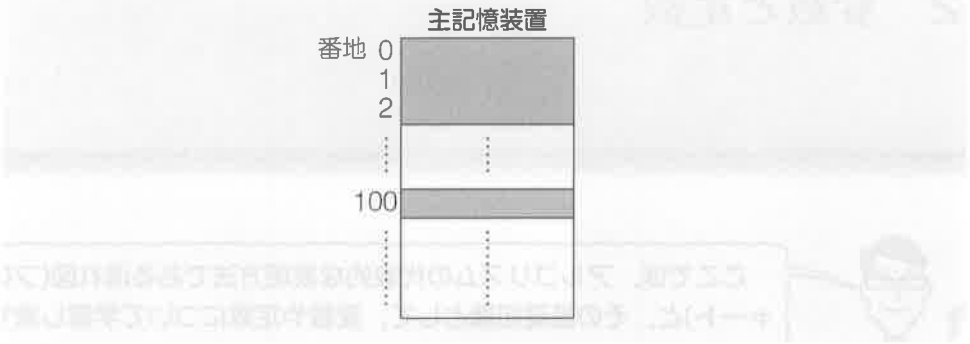
たとえば，上記の図のCOBOLによる記述は，「Xが30 より小さかったら“在庫不足”と表示する」ことを示しています。在庫数が30未満であれば，在庫不足として注意を促すような目的をもつアルゴリズムをCOBOLで表現したものといえます。

データの格納場所と記憶装置

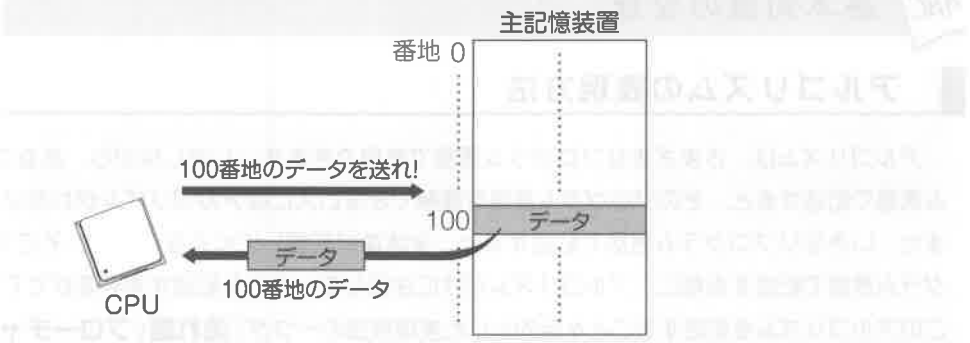
コンピュータに何か作業をさせる場合には，必要なデータを与えておかなければなりません。そのためにはデータを格納する場所がなければなりません。また，処理した後のデータを格納する場所も必要となります。この役割を担うのが記憶装置です。

記憶装置は，半導体メモリで構成される主記憶装置と，ハードディスク装置などの補助記憶装置に分かれます。ここでは，それぞれの装置の詳細には触れませんが，これらの装置にはデータの格納場所を示す“アドレス”（番地ともいう）とよばれるものがつけられているということを理解しておきましょう。

たとえば，主記憶装置には次の図のようにアドレスがつけられています。



コンピュータのCPUは，このアドレスを指定して，主記憶装置からデータを取り出したり，反対にデータを格納したりします。



参考：プログラム言語とデータの格納場所

プログラム言語を用いてプログラミングを行う場合，データの格納場所を考えておかなければなりません。機械語でプログラムを記述する場合には，データの格納場所も“0”と“1”の数字列で示されたアドレスを用いて指定する必要があります。

しかし，CやCOBOLなどの高水準言語では，直接格納場所のアドレスを指定しなくてもよいようになっています。これらの高水準言語には，“変数”(COBOLではデータ項目)などとよばれる，データを格納する入れものが用意されています。この変数をプログラム内で定義すると，翻訳された機械語プログラムを実行する際に，それぞれの変数用の格納場所として，主記憶装置上に特定のアドレスが割り当てられます。こうした変数をプログラム内であらかじめ定義することを変数の“宣言”とよびます。

1-2 変数と定数

ここでは，アルゴリズムの代表的な表現方法である流れ図(フローチャート)と，その基礎知識として，変数や定数について学習します。

基本知識の整理

アルゴリズムの表現方法

アルゴリズムは，さまざまなプログラム言語で表現できます。しかしながら，あるプログラム言語で記述すると，そのプログラム言語を理解できない人にはアルゴリズムがわかりません。また，いきなりプログラム言語で記述すると，全体像が把握しにくくなります。そこで，プログラム言語で記述する前に，アルゴリズムだけに注目して，それを記述する必要がでてきます。このアルゴリズムを記述することを目的とした表現技法の一つが「**流れ図**（**フローチャート**）」です。

変数とは

変数は，データを格納する“箱”のようなものと考えてください。プログラムはこの変数を対象に処理を行うので，データをプログラムで処理する場合には，処理に先立ちデータを変数に格納しなければなりません。アルゴリズムの学習をするために，変数とはどういうものかをしっかりと理解しましょう。

定数とは

定数とは，読んで字のごとく値の定まったものをいいます。数値の5は常に5ですから，すなわち定数です（正しくは**数値定数**とよびます）。また，定数には数値だけでなく文字もあり，これを**文字定数**とよぶことがあります。

理解を深めよう

変数の性質

変数には次にあげるような特徴があります。重要ですから，しっかり覚えましょう。

①変数には名前(変数名)がついている

変数には，名前をつけられます。これを**変数名**とよびます。この変数名によって，どの変数であるか判別されるわけです。変数名は自由につけてかまいませんが，わかりやすくする必要があります。流れ図を記述する場合は特に制約はありませんが，プログラム言語を用いてプログラミングする場合は，そのプログラム言語ごとの制約(文字種類や文字数など)に従わなければなりません。

②変数の中には最初は何が入っているかわからない

プログラムの内部で変数Aを宣言しておく(Aという名前の変数を使用することをコンピュータシステムに知らせる)と，コンピュータシステムはデータを格納する領域をメモリ上に確保します。このとき，確保した領域にどのようなデータが格納されていたかはわかりません。このように，変数にどのような値が入っているかわからない状態を**不定**とよびます。つまり，変数を用意しただけでは，変数Aの**初期値**(最初の値)が0であるとは限らないのです。

この性質は重要です。流れ図を記述する場合に必要な知識ですから，覚えておきましょう。

③変数には，値を入れることができる(代入)

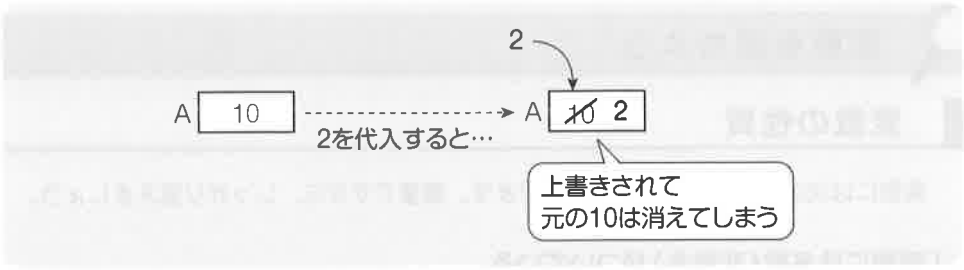
変数に値を入れることを**代入**とよびます。たとえば，流れ図において，変数Aに10(数値定数)を代入するときには，

10 → A

のように記述します。変数には数値・文字などの値を代入することができますが，数値を格納するための変数に文字を代入したり，文字を格納するための変数に数値を代入することはできません。流れ図では特に区別しませんが，プログラミングの際には重要です。

④変数に格納できる値は一つだけである

変数には，同時に一つの値しか格納できません。たとえば，あらかじめ変数Aに10が格納されている場合に，その変数Aに2を代入する場合を考えてみましょう。この代入により，変数Aの内容は10から2に更新され，もともと格納されていた値10は消えてしまいます。

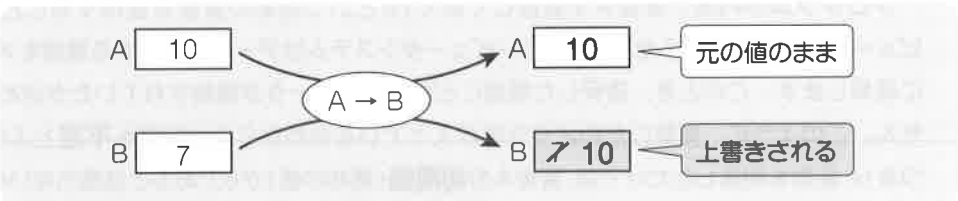


⑤変数には，ほかの変数の値を代入することができる

変数には，ほかの変数の値を代入することができます。たとえば，変数Aの内容を変数Bに代入するときには，

A → B

のように記述します。このとき，変数Aの内容が変数Bにコピーされると考えましょう。移動するわけではないので，変数Aの内容は変わりません（元の値のまま）。



⑥変数には，演算結果を代入することができる

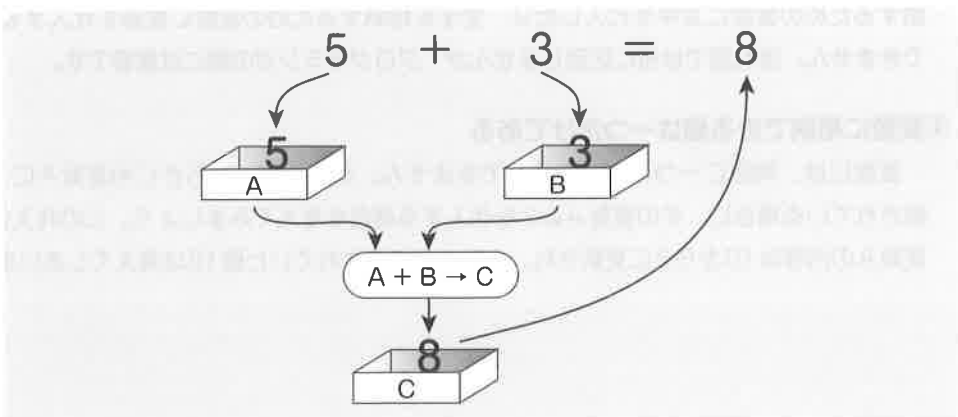
変数には，演算の結果を代入することができます。たとえば，

A ＋ 5 → B

は，「変数Aの内容に5を加えた結果を，変数Bに代入する」ということです。もちろん，変数どうしの演算でもかまいません。この場合は，

A ＋ B → C

のように記述され，「変数Aの内容と変数Bの内容の合計を，変数Cに代入する」という意味になります。たとえば，変数Aに5，変数Bに3が格納されているとすると，「A ＋ B → C」によって変数Cに“5＋3”の結果である「8」が格納されることになるわけです。



⑦自分自身に計算結果を代入する

計算元のデータが格納されている変数に計算結果を代入することもできます。たとえば，

Ａ＋１→Ａ

のように記述することができるわけです。これは，「変数Ａの値に１を加えた結果を，改めて変数Ａに代入する」という意味であり，結果として「変数Ａの値を１増加させる」ことになります。もちろん，この性質は変数どうしの演算にも適用できます。たとえば，

Ａ＋Ｂ→Ａ

と記述すれば，「変数Ａの内容と変数Ｂの内容の合計を，変数Ａに代入する」ということになります。

文字定数と変数名の区別

流れ図を書く場合，ある数値や文字が定数を意味するのか変数名であるのかを区別できなければなりません。たとえば，単に

Ａ

とあった場合，Ａという変数名の変数を意味しているのか，文字のデータである文字定数のＡなのかわかりません。そこで，文字定数を記述する場合は，

“Ａ”

というように，ダブルクォーテーション(“”)でくくります。ですから，単に

Ａ

とある場合は，変数Ａ(変数名Ａの変数)を意味し，

“Ａ”

とある場合には，文字データのＡであることになります。

さらに，“10”と記述する場合があります。これは，文字データとしての“10”（文字定数）を意味します。単に

10

とある場合は，数値データとしての10(数値定数)を意味します。数値データは数値の演算に使えますが，文字データは数値の演算には使えません。つまり，５＋３を計算して結果の８を変数Ａに格納したい場合は，

５＋３→Ａ

と記述します。“5”，“3”のようにダブルクォーテーションでくくると，数値計算の対象データとはみなされないわけです。

なお，“10”や10はそれぞれ文字定数，数値定数を意味しますから，数字だけを使って変数名とすることはできないことを理解しておいてください。ただし，

Ａ10

というような変数名は使うことができます。

1-3 流れ図の記号と処理構造 その1

～順次と分岐～

アルゴリズムの代表的な表現方法である流れ図(フローチャート)で用いる記号について学習します。記号の中には，具体的なアクションを記述しますが，日本語などの文で処理内容を記述することもあります。

基本知識の整理

流れ図の記号には，**データ記号**，**処理記号**，**判断記号**，**繰返し記号**などがあります。

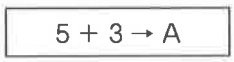
基本処理記号とは

処理を記述する場合の記号です。長方形の中にそこで行う処理の内容を記述します。なお，次に説明する判断記号やループ端記号も処理記号に含まれますが，ここでは，単に処理記号と記述した場合は，基本処理記号を指すものと考えてください。

たとえば，ここまで5＋3の結果を変数Aに格納する場合は，

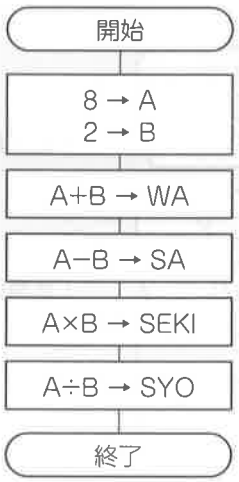
5 ＋ 3 → A

と記述してきましたが，流れ図では処理記号の長方形の中に記述します。



基本処理記号のみで記述されたアルゴリズム

次の流れ図は，変数Aの内容と変数Bの内容による四則演算（足し算，引き算，掛け算，割り算）の結果を，それぞれ和（足し算の答え）をWA，差（引き算の答え）をSA，積（掛け算の答え）をSEKI，商（割り算の答え）をSYOという変数に格納するアルゴリズムを表したものです。



流れ図に記述された処理は，原則として流れ図の上から下へ順に処理されていくと考えてください。一つの基本処理記号の中に，複数の処理を記述することもできますが，この場合も上に記述された処理が先に行われるものと考えてください。このような，制御構造を“**順次**”といいます。

よって，この流れ図では，まず和が変数WAに求められ，次に差が変数SAに求められることになります。そして，すべての処理を終えたときには，WAには10，SAには6，SEKIには16，SYOには4が格納されていることになります。

● 関連用語

端子

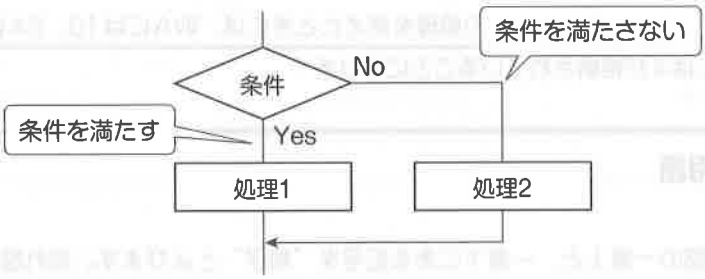
流れ図の一番上と，一番下にある記号を“端子”とよびます。流れ図は開始の端子で始まり，終了の端子で終わることになります。

(基本)データ記号

データ記号は，データの入出力を記述するための記号です。平行四辺形の中に入出力内容を記述します。たとえば，変数Aと変数Bにデータを入力して(キーボードなどから)，AとBの内容の和を変数Cに格納して，そのCの内容を出力する(ディスプレイに表示するなど)流れ図は，次の図のように記述することができます。

判断記号とは

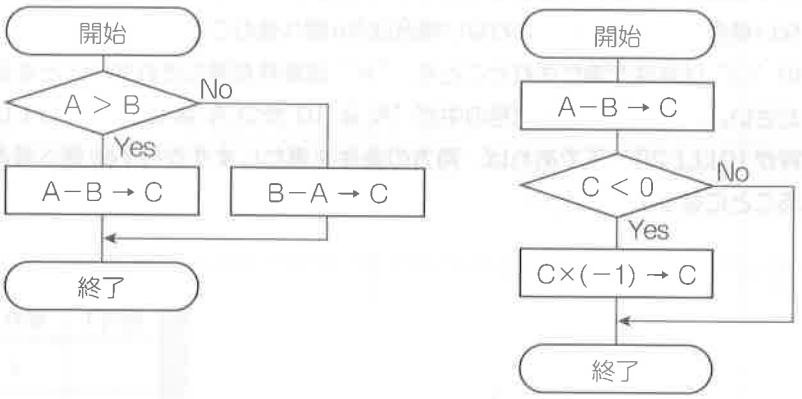
判断記号は“ひし形”をしています。判断記号は，処理を分ける場合に用います。この処理を分けることを“**分岐(選択)**”といい，分けるための判断基準を“条件”といいます。つまり，流れ図の処理は上から順に行われますが，この判断記号があると，そこで処理が分かれるわけです。



上の図の例では，判断記号の中に記述された条件を満たしている場合には，Yesの方へ進み，処理1が実行されます。このときには処理2がある方は通りませんから，処理2は実行されないわけです。逆に，判断記号の中に記述された条件を満たさない場合には，Noの方へ進み，処理2が実行され，処理1は実行されません。

判断記号を用いた簡単な流れ図

次の図の二つの流れ図は，いずれも変数Ａと変数Ｂに格納された値の“差の絶対値”を変数Ｃに求めるアルゴリズムを表現したものです。絶対値とは，正や負の符号を取ったものと考えてください。ですから，“2”と“－2”の絶対値は，ともに“2”となります。



差の絶対値を求める流れ図

あらかじめ変数Ａに5が，変数Ｂに3が格納されているものとしましょう。

左の流れ図では，まずＡとＢの内容を比較しています。この大小の比較が“条件”です。Ａのほうが Ｂより大きい場合，“A ＞ B”は正しいわけですから，条件を満たすことになります。この場合，Yes側へ進み，“A－B → C”が実行され，変数Ｃの内容は2となります。

一方，右の流れ図では，まず“A－B → C”を実行します。この時点で変数Ｃの内容は2となります。この状態で，判断記号で変数Ｃの内容と0とが比較されます。変数Ｃは2ですから，“C ＜ 0”つまり“2 ＜ 0”は正しくありません。よって，No側へ進み処理を終えます。Yes側へは進みませんから，Ｃの内容は2で確定します。

それでは，あらかじめ変数Ａに3，変数Ｂに5が格納されている場合はどうでしょう。

左の流れ図では，ＡとＢの内容を比較します。ＢのほうがＡより大きい場合，“A＞ B”は正しくありませんから条件を満たしません。よって，No側へ進み，“B－A→ C”が実行され，変数Ｃの内容は2となります。

一方，右の流れ図では，“A－B →C”を実行し，変数Ｃの内容は－2となります。この状態で，判断記号で変数Ｃの内容と0とが比較されます。変数Ｃは－2ですから，“C ＜ 0”つまり“－2 ＜ 0”は正しいので，Yes側へ進み，“C×(－1) → C”が実行されます。この処理が行われる前，Ｃは－2ですから，“(－2)×(－1) → C”つまり，変数Ｃには2が格納されることになるわけです。

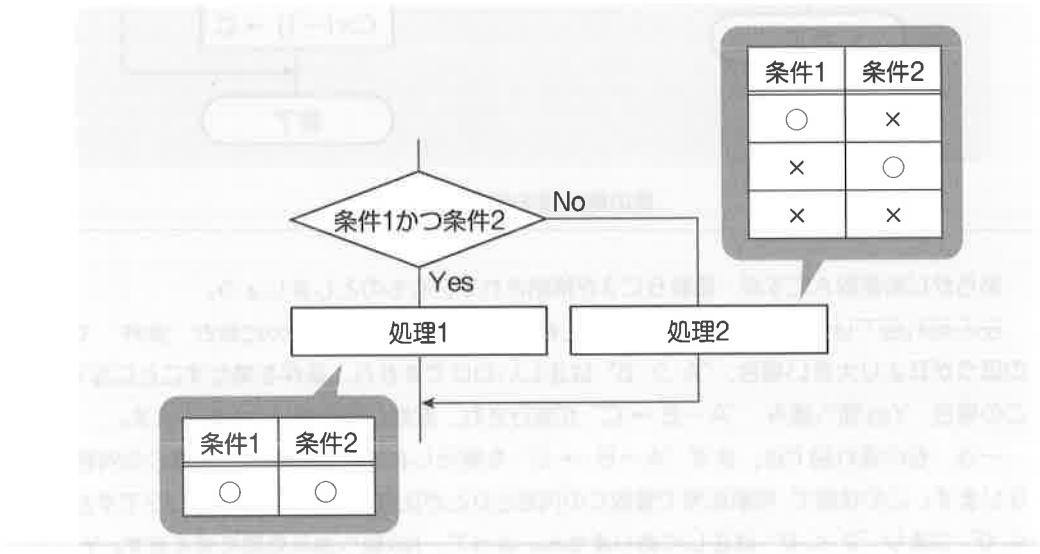
二つの流れ図とも二つの変数の差の絶対値が求められることがわかるでしょう。「ある答えを求めるためのアルゴリズムは複数存在することがある」ということを忘れないでください。

理解を深めよう

複合条件

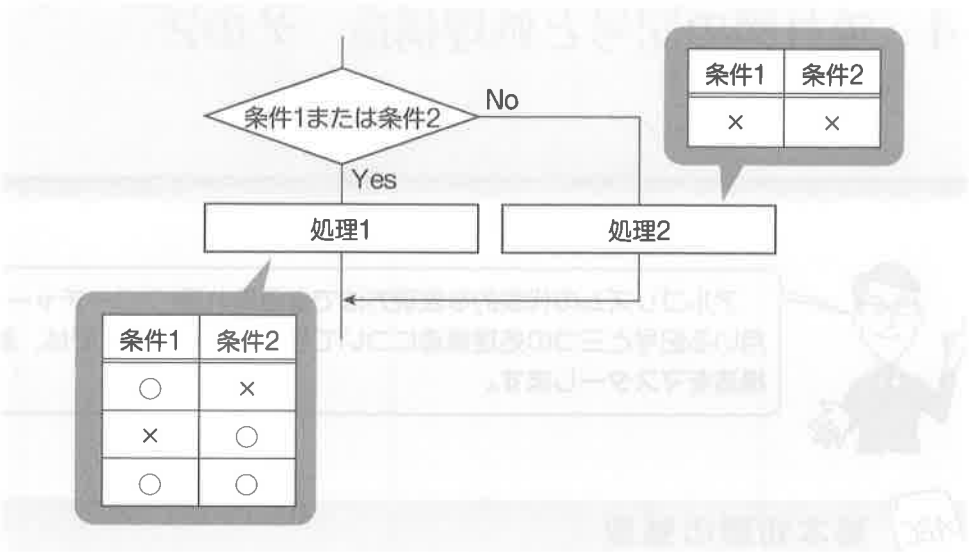
複数の条件を“かつ”や“または”で結んだものを**複合条件**とよびます。この“かつ”は次の図のように条件1と条件2がともに満たされたときYes側に進み，どちらか一方の条件が満たされない場合や，ともに満たされない場合はNo側へ進むことになります。

図中の“○”は条件が満たされたことを，“×”は条件が満たされないことを示していると考えてください。たとえば，判断記号の中が“A ≧ 10 かつ A ≦ 20”となっていた場合，変数Aの内容が10以上20以下であれば，両方の条件を満たしますからYes側へ進み，処理1が実行されることになります。



“かつ”で結ばれた条件

次に“または”を考えてみましょう。この“または”は次の図のように条件1と条件2のどちらか一方，あるいは両方が満たされたときYes側に進み，両方の条件が満たされない場合はNo側へ進むことになります。たとえば，判断記号の中が“A < 10 または A > 20”となっていた場合，変数Aの内容が10より小さいか，あるいは20より大きければ，どちらか一方の条件を満たしますからYes側へ進み，処理1が実行されることになります。

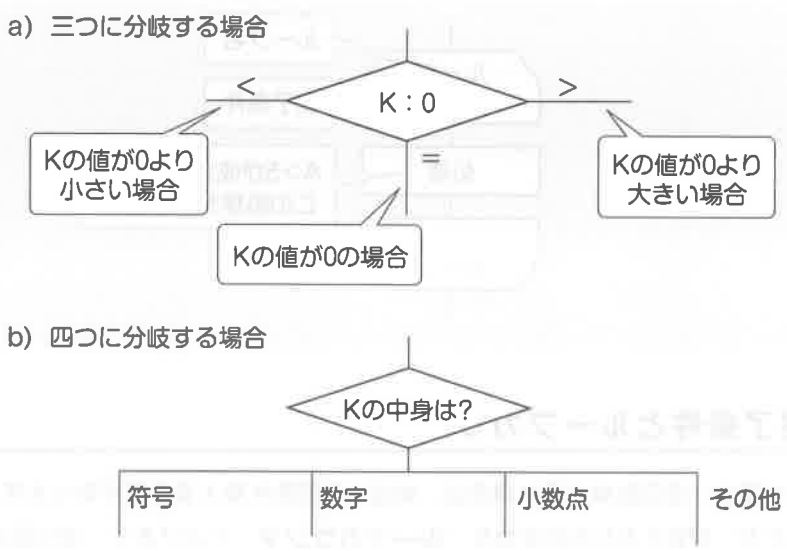


“または”で結ばれた条件

なお，“かつ”を“AND”，“または”を“OR”と記述することもあります。

多分岐

ここまで説明してきた分岐は，処理が二つに分かれる構造ですから“**二分岐**”とよびます。これに対して三つ以上に分岐先が分かれる分岐の構造を“**多分岐**”とよびます。多分岐の構造は，やはり判断記号を用いて次の図のように記述します。



多分岐の構造

1-4 流れ図の記号と処理構造 その2

～繰返し～

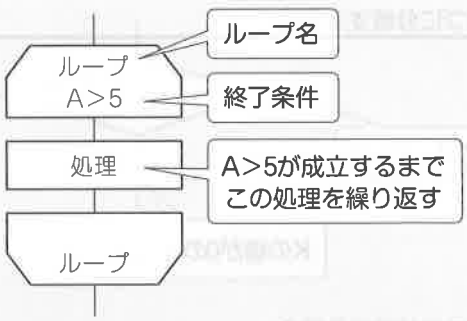
アルゴリズムの代表的な表現方法である流れ図(フローチャート)で用いる記号と三つの処理構造について学習します。ここでは，繰返し構造をマスターします。

基本知識の整理

ループ端記号とは

コンピュータの処理では，同じ処理を繰り返す場面が数多く登場します。そのような**繰返し**（**ループ**ともいう）処理を表現する記号が**ループ端記号**です。次の図の中の一番上と一番下の記号がループ端記号です。前者をループ始端，後者をループ終端とよびます。このループ端記号で挟まれた処理を，ループ端記号に記述された条件（**終了条件**）を満たすまで繰り返すことになります。条件を満たさない限り，ループ終端まで行ったらループ始端に戻ってくることになります。

なお，終了条件には“かつ”や“または”で結んだ，複合条件を用いることもできます。



終了条件とループカウンタ

同じ処理を一定回数繰り返す場合は，繰返しの回数を数える必要があります。この“何回繰り返したか”を数えるための変数を“**ループカウンタ**”とよびます。次の図は処理Ａを10回繰り返す処理の流れ図です。右側の流れ図では，処理Ａを10個記述しています。これでもよいわけですが，左の流れ図のようにループ端記号を用いて繰返しの構造で記述すると，流れ図を短く，わかりやすく記述することができます。

左の流れ図では，Cntという変数をループカウンタとして用いています。ループカウンタはループ内で処理Aを1回実行した直後に1加算されています。ところで，ループに入る前にループカウンタCntに1を代入しています。これは，変数の内容が最初は不定であるからです。このように，変数（ここではループカウンタ）の内容をアルゴリズムの目的に合致するようにあらかじめ値を設定することを“**初期化**”とよびます。

さて，1に設定されたCntは，ループ（繰返し）処理の1回目で，2となります。2回目では3となり，3回目では4となります。このように考えていくと，Cntの内容が次のように変化していくことがわかります。

ループカウンタCntの内容の変化

	Cntの値	処理Aの実行回数
1回目の終了条件判定（ループ開始直前）	1	0回
2回目の終了条件判定（ループ1回目終了直後）	2	1回
3回目の終了条件判定	3	2回
4回目の終了条件判定	4	3回
5回目の終了条件判定	5	4回
6回目の終了条件判定	6	5回
7回目の終了条件判定	7	6回
8回目の終了条件判定	8	7回
9回目の終了条件判定	9	8回
10回目の終了条件判定	10	9回
11回目の終了条件判定（ループ10回目終了直後）	11	10回

ここで，10回目の終了条件判定を考えてみましょう。その直前，ループの9回目で，ループカウンタCntの内容は9から10になります。そして，処理Aは9回実行し終えたことになります。あと1回実行する必要があります。ですから，終了条件が“Cnt ＞ 10”となっているのです。Cntの内容が10では，まだこの条件は満たされませんから，もう一度，ループ内の処理Aを行います（ループ10回目）。そして，ループ10回目を終えてループ始端に戻ってきたとき，Cntは11になっていますから，終了条件を満たし，ループ終端の次へ処理が移るわけです。

● **関連用語**

基本制御構造

ここまでに取り上げた“順次”，“分岐（選択）”，“繰返し（ループ）”の三つの制御構造のことを，**基本制御構造**といいます。

理解を深めよう

前判定型と後判定型

ループ始端に終了条件が記述されているものを“**前判定型**”，ループ終端に終了条件が記述されているものを“**後判定型**”とよびます。前判定型では，

判断 → 処理 → 判断 → 処理 → … → 判断（成立） → ループ終了

という流れとなります。つまり，終了条件の判断を行った後に処理を行うような繰返しとなるわけです。逆に後判定型では，

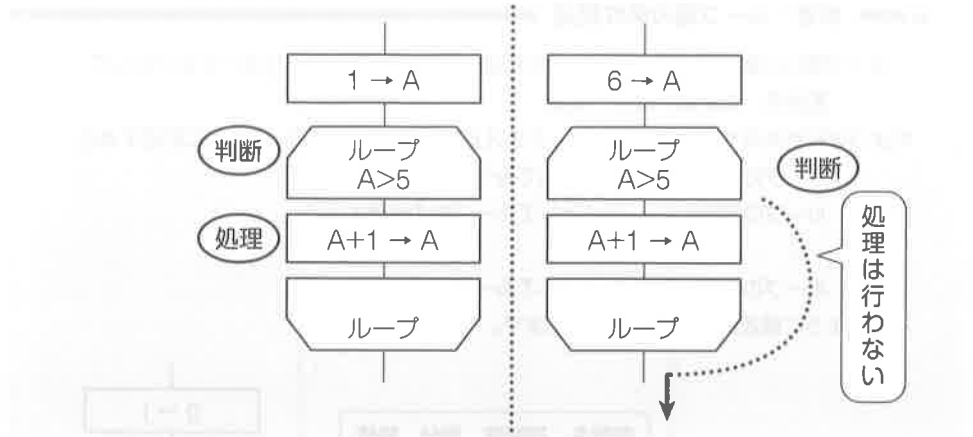
処理 → 判断 → 処理 → 判断 → … → 判断（成立） → ループ終了

という流れとなります。つまり，処理を行った後に終了条件の判断を行うような繰返しとなるわけです。

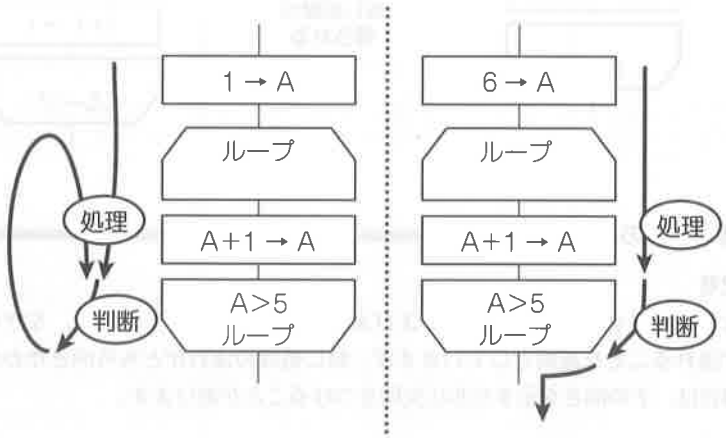
両者の違いは，ループに入る前にすでに終了条件を満たしているような場合に現れます。前判定型の繰返しでは，次の図の右の流れ図のように，ループに入る前に終了条件を満たしている場合，ループ内部の処理

A＋1 → A

は1回も実行されません。



ところが，後判定型の繰返しでは，仮にループに入る前に終了条件を満たしている場合でも，少なくとも1回はループ内の処理を行うことになります。



参考：ループ端の条件記述

ループ端には基本的に終了条件のみを記述しますが，表記のバリエーションとして，

変数名：初期値，増分，終値

のような記述を許すこともあります。たとえば，“i：0，1，5”のように記述すると，

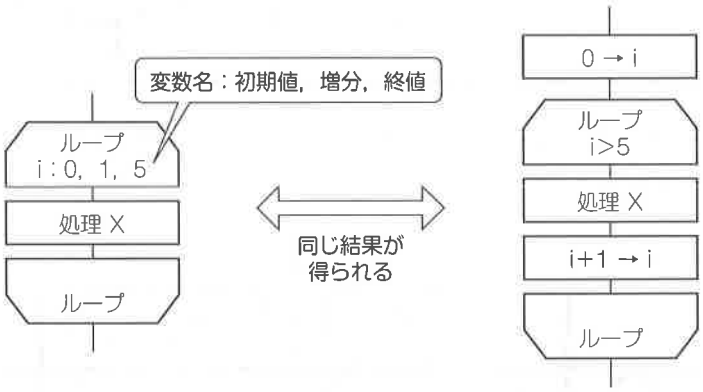
ループの1回目 … i=0としてループ内の処理を行う

ループの2回目 … i=1としてループ内の処理を行う

⋮

ループの6回目 … i=5としてループ内の処理を行う

というように繰返しが行われて終了します。



参考：その他の記号

● 線記号

処理などの記号を結ぶ線を線記号とよびます。流れ図では，上から下，左から右に向かって処理が流れることを原則としていますが，特に処理の流れがどちら向きかわからなくなるような場合は，その向きを示すために矢印をつけることがあります。

● 結合子

線が交差したり，図が大きくなり複数のページにまたがって記述しなければならない場合に用いる記号です。結合子の中には数字などの識別できる名前を記述しておきます。同じ名前の結合子は線がつながっているのと同じと考えることになっています。

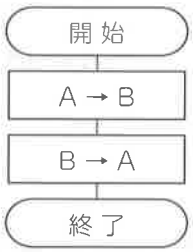
1-5　変数どうしの内容の交換

変数に格納された内容(データ)を交換する処理は頻繁に登場します。基礎中の基礎ですから正しく理解しましょう。

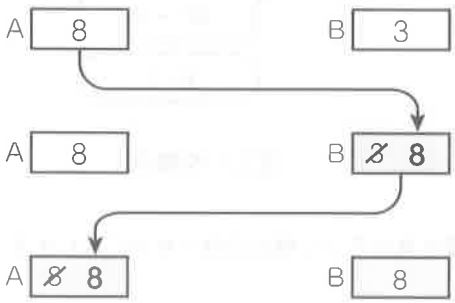
基本知識の整理

変数の性質と誤った交換の方法

変数に格納できる値は一つだけです。ですから，ある値を格納している変数に新たに値を格納すると，元の値は上書きされてしまいます。それを念頭に，右の流れ図を見てください。



この流れ図では，変数どうしの中身を正しく交換することができません。なぜなら，最初に行われる“A → B”で変数Aの内容が変数Bにコピーされ，Bの元の内容が失われてしまうからです。次の図はAの元の内容が8，Bの元の内容が3であった場合に上の流れ図を実行した場合の様子を示しています。結果として，A，Bの内容はともに元のAの内容である8になってしまうのです。もちろん，流れ図の処理の順序を入れ替えてもうまくいきません。その場合は，ともに元のBの内容である3になってしまいます。

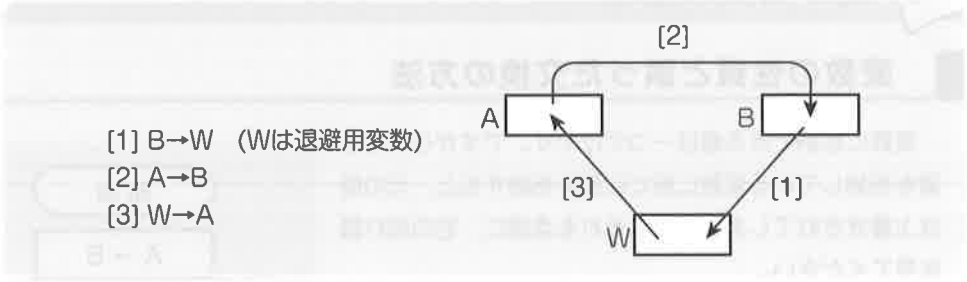


理解を深めよう

正しい交換の方法

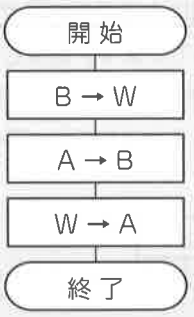
先の手順ではどうしてうまくいかないのでしょう。それは,「変数に格納できる値は一つだけ」だからです。そこで，もう一つ別の変数を用意してあげることにしましょう。ここではそれを変数Wとします。

変数Aの内容をいきなり変数Bにコピーするのではなく，元のBの内容がなくならないように「あらかじめBの内容をWにコピーしておき」，その後で「Aの内容をBにコピーし，次にWからAにコピーする」という方法をとればよいのです。このときの変数Wは,変数Bの内容を“**退避**”しておくために用いられているので，“**退避用変数**”あるいは“**作業用変数**”とよばれます(ワーク変数，または単にワークとよぶこともあります)。



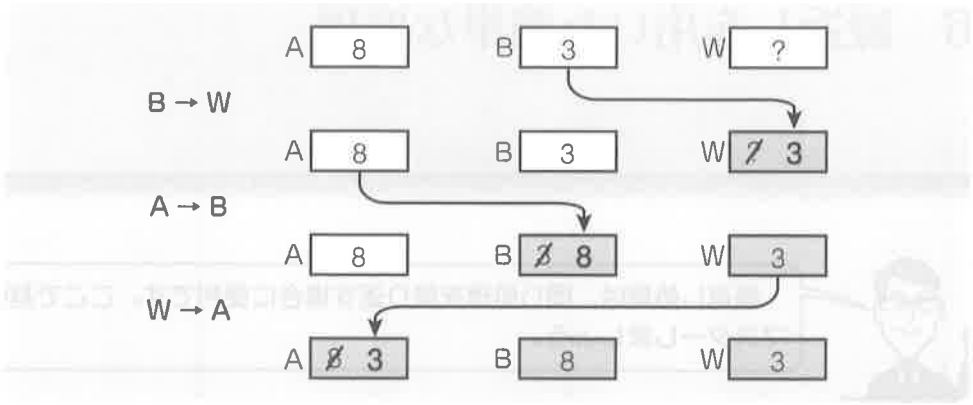
退避用変数Wを用いた交換のイメージ

上記の手順を流れ図で示すと次のようになります。



正しい交換の流れ図

流れ図に従って処理を進めていく場合の様子を次に示します。



正しい交換の様子

今度はうまくＡとＢの内容を交換できました。なお，このとき，退避用変数Ｗの最初の中身は処理結果に影響しませんから，特に値を定めておく必要はありません（図中の？は変数の内容が不定であることを示しています）。

さて，上記の例では，はじめに変数Ｂの内容を退避用変数Ｗに退避しました。もちろん，変数Ａの内容を退避用変数Ｗに退避しても正しく交換を行うことができます。

その場合の処理順序は，

① Ａ → Ｗ
② Ｂ → Ａ
③ Ｗ → Ｂ

となります。本当にうまくいくか，図を書いて確かめてみるとよいでしょう。

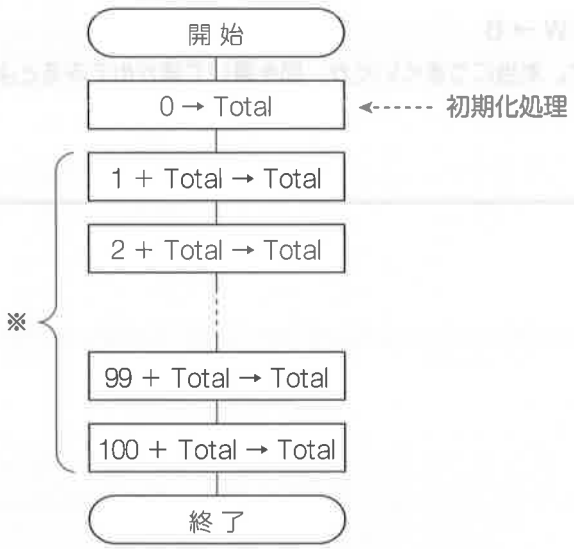
1-6 繰返しを用いた簡単な処理

繰返し処理は，同じ処理を繰り返す場合に便利です。ここで基礎をマスターしましょう。

基本知識の整理

1から100までの整数の合計を求める

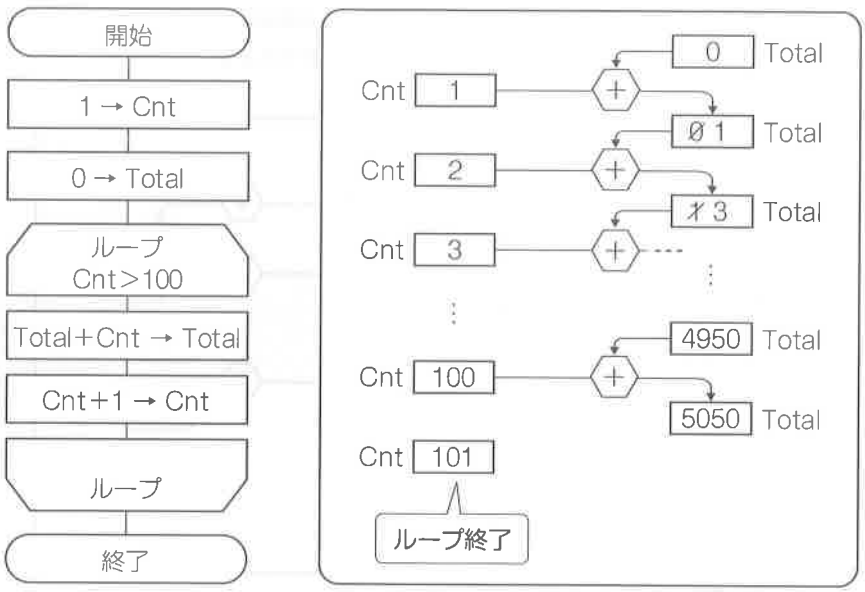
1から100までの整数の合計を求める場合，どのように行えばよいでしょうか。アルゴリズムをはじめて学習する方が思いつくのは，次のような流れ図でしょうか。この流れ図では変数Totalに合計を求めています。



1から100までの整数の合計を求める流れ図（その1）

しかし，この流れ図では同じような処理を100個も並べて記述しなければなりません。また，後になって「1から100までではなく，1から1000までの合計を求めたい」という要求が出たときに，その変更は容易ではありません。

繰返しを用いれば，次のように「スマートに」アルゴリズムを記述することができます。



1から100までの整数の合計を求める流れ図(その2)

このようにすることで，流れ図を簡潔に示すことができます。

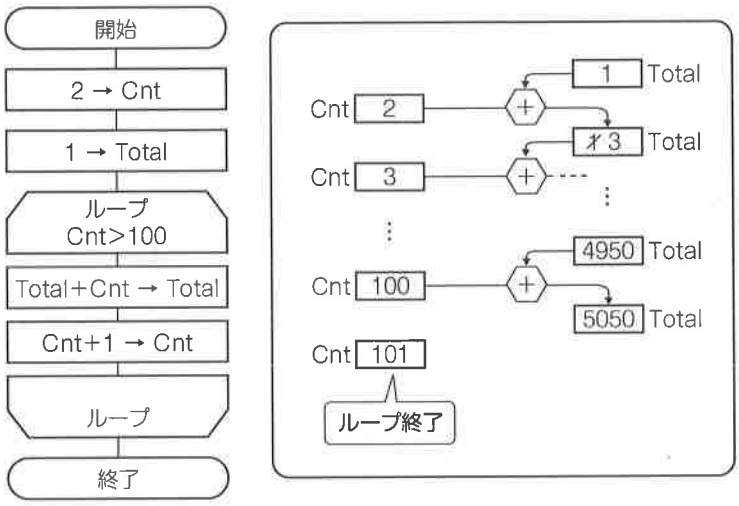
また，ループの繰返しの終了条件を変更するだけで，1からnまでの整数の合計を求める流れ図に変えることができるのです。たとえば，1から1000までの整数の合計を求めるように変更するには，ループの終了条件を

Cnt ＞ 1000

とすればよいわけです。

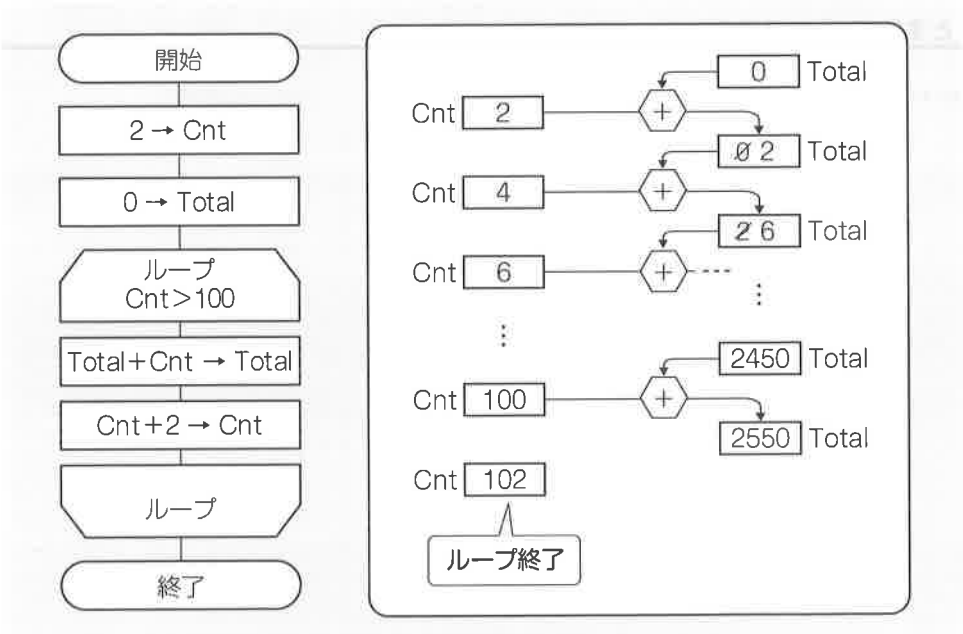
参考：初期値と繰返し回数

前出の流れ図では，変数Totalの初期値を0とし，ループを100回繰り返して，1から100までの整数の合計を求めています。しかし，次のようにTotalの初期値を1，ループカウンタCntの初期値を2とすることで，ループの繰返し回数を1回減らすことができます。



1から100のうちの偶数の合計を求める

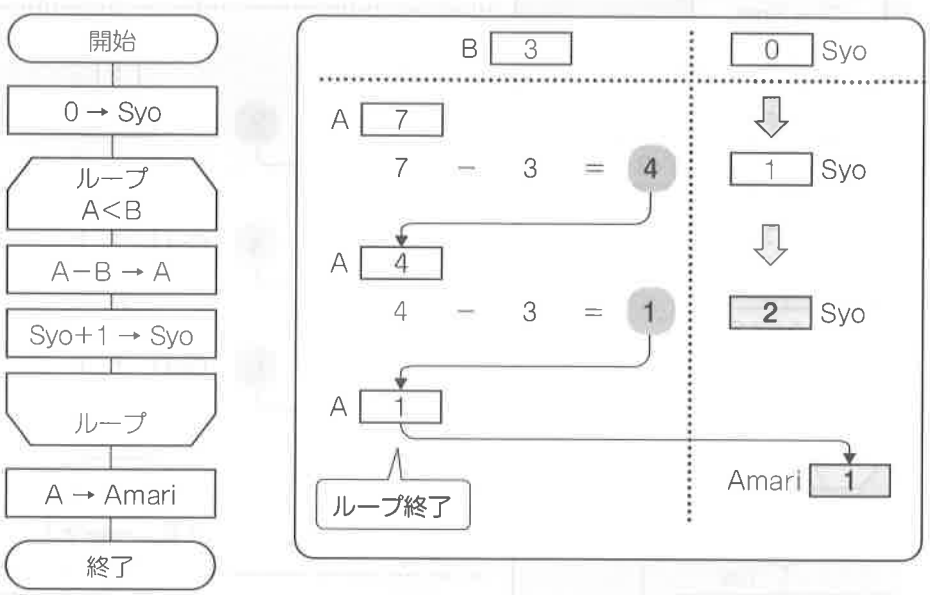
少し変更して，「1から100のうちの偶数の合計を求める流れ図」を考えてみましょう。ループカウンタCntは，1から一つずつ増やしていましたが，これを「2から2ずつ増やす」ように変更すればよいわけです。流れ図は次のようになります。



1から100のうちの偶数の合計を求める流れ図

割り算の答えを引き算の繰返しで求める

割り算の答えを求めるには，もちろん割り算を行えばよいのですが，割り算を使わなくても引き算の繰返しで答えを得ることができます。たとえば，7÷3の答えは2あまり1ですが，これは7から3を引いていき，引ききれなくなったときに「引くことができた回数」と「残りの数」にほかなりません。ここで，7が変数Aに，3が変数Bに格納されているものとして，次の図を見てください。



ループの1回目では，AからBを引いてそれを再びAに格納しています。そのとき，Syoの値を1増やしています。これは，“1回引けた”ということを意味します。そして，ループの先頭に戻り，終了条件をチェックします。終了条件の

A ＜ B

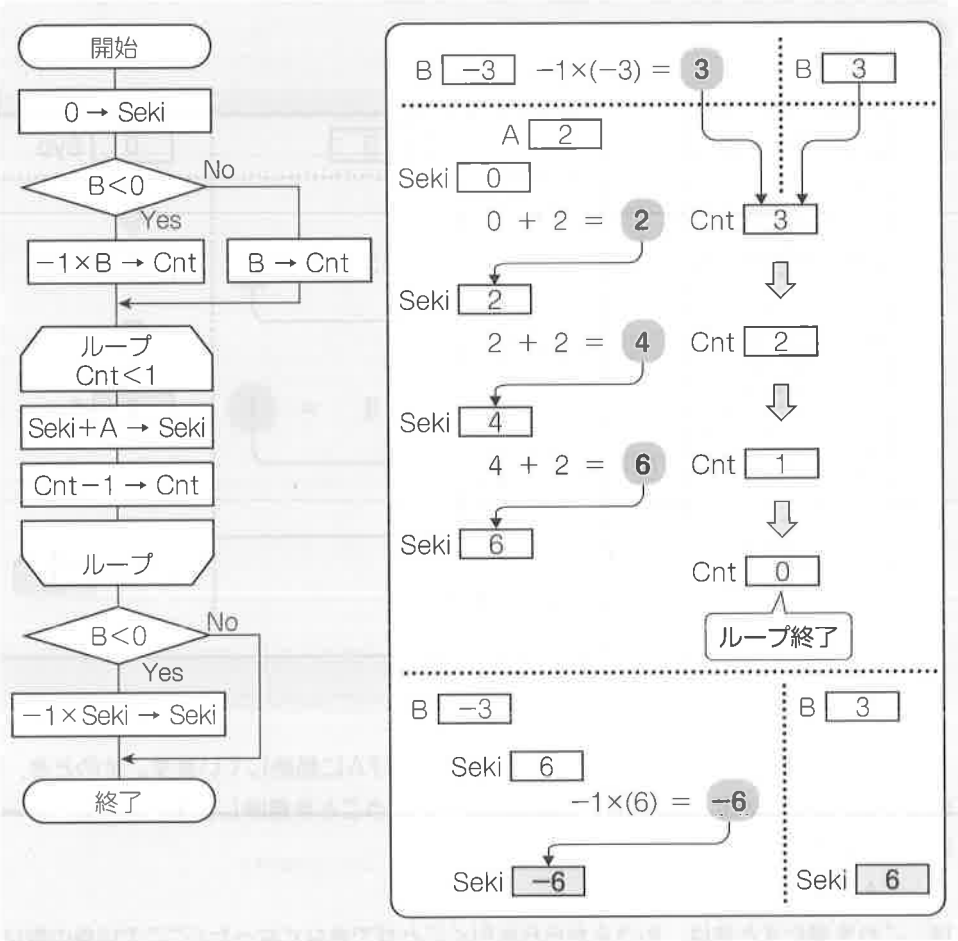
は，これを満たすときは，もうAからBを引くことができなくなった（ここでは負の数は考えません）ことを意味します。ループの2回目ではAは4ですから，繰返し処理を続行します。今度は，Aの値は1になり，Syoは2となります。ループの3回目を行おうとしたとき，終了条件を満たしますから，ループ内の処理は行わず，

A → Amari

として，Aに残っていた1を変数Amariに格納して処理を終了します。このようにして割り算を使わずに，A÷Bの結果である“1あまり2”を変数Syoと変数Amariに求めることができたわけです。ほかの値でも正しく処理されることを確かめてみてください。

掛け算の答えを足し算の繰返しで求める

今度は掛け算です。掛け算は足し算の繰返しで答えを得ることができます。ここでは正の数だけでなく，負の数の掛け算もできるような流れ図としています。図では，変数Aに2，変数Bには－3が入っていた場合と3が入っていた場合を並べて示しています。



まず，変数Bに－3が格納されていた場合を考えてみましょう。流れ図の最初の処理は，掛け算の答えを求めるための変数であるSekiの内容を0で初期化しています。続く分岐処理では，Bの内容が正であるか負であるかを判定し，負である場合はBの内容を－1倍してから，Cntへ格納します。正の場合は，そのままCntに格納します。さて，この時点でBの内容が－3であっても3であってもCntには3が格納されることがわかります。そうした上で，繰返し処理を行うわけです。

ループ処理の内部では，まずSekiにAを加算してSekiに格納します。そして，Cntの内容を一つ減らしてループの先頭に戻ります。ループの先頭では，終了条件をチェックしますが，Cntは2ですから続行です。あとは，終了条件を満たすまで処理を繰り返すと，図の右にあるように，ループ終了時点では，

Aは2
Sekiは6
Cntは0

となっています。さて，ここでBが－3(負)の場合は掛け算の結果は－6となっていなければなりません。そこで，Bが負であるか否かをチェックして，負である場合にはSekiの内容の符号を変えるために，“B ＜ 0”による処理の分岐を行っているのです。

もちろん，正の数のみしか処理しないのであれば，流れ図中のループの前の処理は単純になりますし，ループの後の処理は必要ありません。

参考：初期化が必要な場合

「変数の内容は，最初は**不定**なので，**初期化**が必要」と記述してきましたが，常に初期化が必要というわけではありません。たとえば，3×2の値を変数Aに格納する処理であれば，

3×2 → A

とすればよく，この処理の前にAを初期化しておく必要はありません。では，どのような場合に初期化が必要になるのでしょうか。それは，ループ内で**足し込み処理**を行う場合などです。足し込み処理とは，

Seki ＋ A → Seki

や

Cnt － 1 → Cnt

のように，ある変数に値を加えたり引いたりしたのち，その変数自身に結果を格納しなおすような処理をいいます。このような処理では，最初のSekiやCntの値が不定であると，最後の結果が正しく求められないからです。ですから，このような場合は，ループに入る前に初期値を設定する初期化処理が必要になるのです。

1-7 配列と繰返し処理

配列と繰返し処理は，アルゴリズムを勉強していく上で非常に重要なテーマです。なぜ配列が必要なのか，配列のメリットは何かについて考えていきましょう。

基本知識の整理

変数の弱点と配列

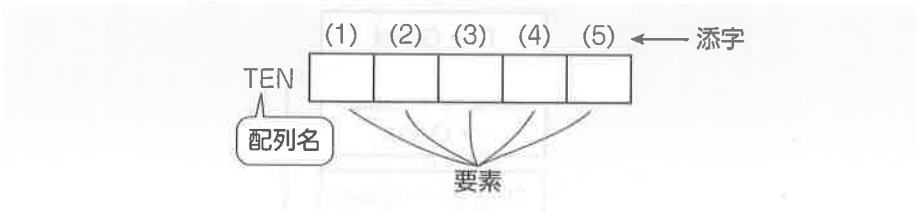
変数には一つの値しか格納できません。ですから，複数のデータを格納する場合にはデータの数だけ変数を用意する必要があります。たとえば，学生のテストの成績の平均点を求める場合を考えてみましょう。安藤さん，加藤さん，佐々木さん，森さん，和田さんという5人の学生がいたならば，区別がつくように，

Ando，Kato，Sasaki，Mori，Wada

というような変数に得点を格納しておきます。さらに，合計点を格納するGokeiと平均点を格納するHeikinという変数を用意して，次のような流れ図を書くことになります。

この場合は5人ですからまだよいのですが，人数が増えるとそれだけ変数の数は多くなり，またGokeiへの足し込みを繰り返しているのにループを使うこともできず，流れ図も長くなってしまいます。そこで，用意されるものが**配列**です。配列は「同じ変数名をもつ同じ型の変数の集まり」です。上の得点のように同じ種類のデータを格納するような場合に用いることができます。

「同じ変数名」と聞いて「どうやって区別するの？」と困惑された方もおられると思います。配列ではそれぞれに「**添字**(そえじ)」とよばれる番号をつけて区別しています。たとえば，5人の得点を格納するためには，下の図のようなイメージで配列を作成します。



配列のイメージ

図の配列は，配列名がTEN，要素数が5となります。**要素**とは，配列を構成する個々の変数のことです。それぞれの要素は添字によって区別され，

TEN(1)，TEN(2)，TEN(3)，TEN(4)，TEN(5)

のように表現されます。個々の要素の性質は，変数の性質とまったく同じと考えてかまいません。

また，添字は上のように数値定数で表してもよいですし，変数を用いることもできます。たとえば，添字として変数iを用いることにすれば，

TEN(i)

と表現することもできます。この場合，添字である変数iの値が3であれば，

TEN(3)

を，変数iの値が1であれば，

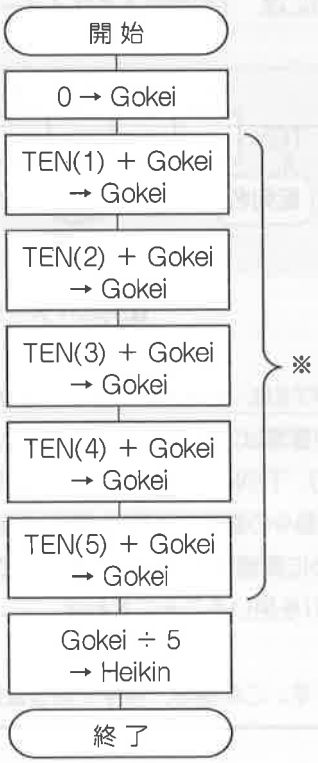
TEN(1)

を意味することになります。

理解を深めよう

配列を使った平均点を求めるための流れ図

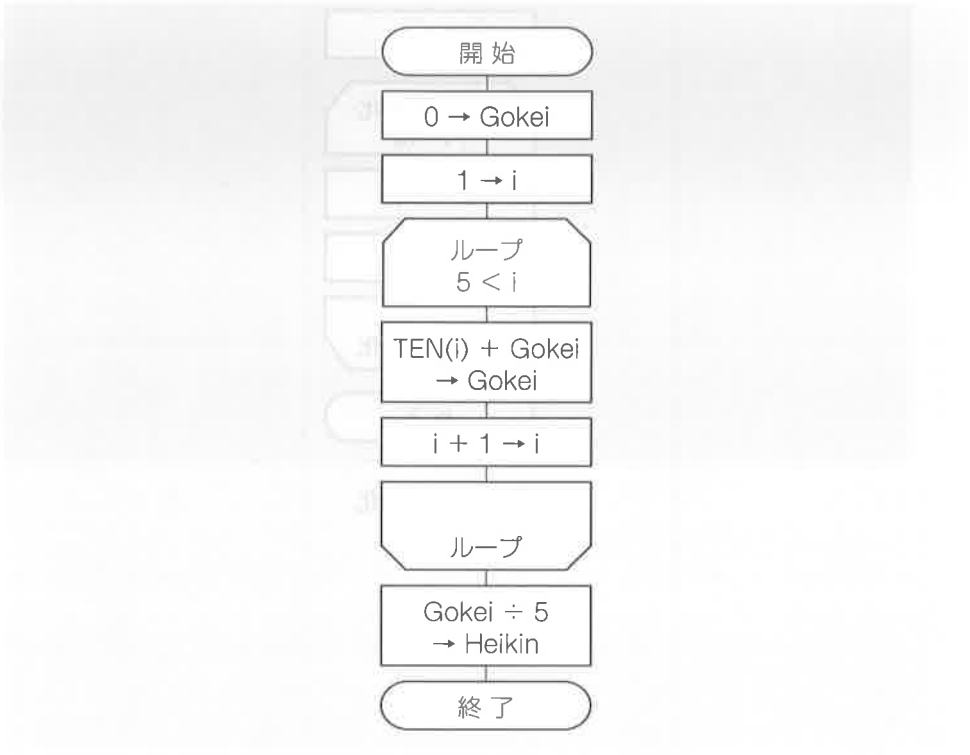
配列を使って前出の流れ図を書き換えることにしましょう。5人の得点を配列TENの各要素に格納してあるものとすると次のようになります。



配列と繰返し処理

前の流れ図の※印の部分に注目してください。配列TENの添字以外はすべて同じ記述になっています。しかも，その添字の値は1ずつ増加しています。ですから，※部分を繰返し処理とすればもっと簡潔に流れ図を記述できるはずです。その際，添字をループカウンタとして扱えばよいのです。

ループカウンタ及び添字として用いる変数を変数iとすると，流れ図は次のようになります。

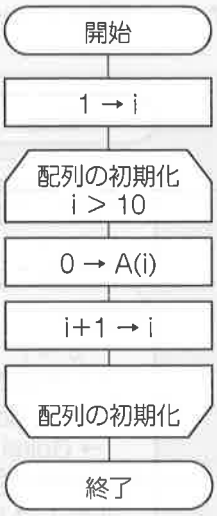


繰返し処理を導入した平均点を求める流れ図

このように配列を使うことで，繰返し処理を用いることができるようになる場合もあります。配列と繰返し処理はアルゴリズムの中でも基本中の基本ですから，しっかり理解するようにしてください。

配列の初期化

配列の各要素は変数と同様に，定義した段階ではその内容は不定です。ですから，場合によっては初期化処理を行う必要があります。次の流れ図は，10個の要素をもつ配列Aの各要素を0で初期化するためのものです。このような場合も，繰返し処理で表現すると効率的です。



配列の初期化

1-15 2次元配列

添字が一つの配列を“1次元配列”とよびます。1次元配列は直線的なイメージです。これに対して，添字二つで要素を特定する配列を“2次元配列”とよびます。

基本知識の整理

2次元配列とは

2次元配列の配列のイメージは次の図のようになります。一般に縦方向を“行”，横方向を“列”とよび，それぞれを添字で指定することで，特定の要素が選ばれます。この時，行方向の添字を**第1添字**，列方向の添字を**第2添字**とよびます。

配列名（第1添字，第2添字）

なお，行の数をM，列の数をNである2次元配列をM行N列の2次元配列とよびます。ですから，図の2次元配列は，3行5列の2次元配列ということになります。