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，2023年春からの通年化に伴い，ITを活用したサービス，製品，システム及びソフトウェアを作る人材に必要な基本的知識・技能をもち，実践的な活用能力を身に付けた者を対象とした試験となっています。そのため，現代のIT社会に必要な幅広い知識が問われています。

本書は，基本情報技術者試験の科目B試験の中核となる「アルゴリズムとプログラミング」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。

TAC情報処理講座

アルゴリズム　目次

Part1　アルゴリズムの基礎 1
1－1　アルゴリズムとは何か 2
1－2　変数と定数 6
1－3　基本制御構造　その1　～ 順次と分岐 ～ 13
1－4　変数どうしの内容の交換 21
1－5　基本制御構造　その2　～ 繰返し ～ 24
1－6　繰返しを用いた簡単な処理 32
1－7　引数と返却値 35
1－8　配列と繰返し処理 39
1－9　二次元配列 45
1－10　計算量 49

Part2　基本アルゴリズム 51
2－1　最大値・最小値を求めるアルゴリズム 52
2－2　基本アルゴリズム(探索)　その1　～ 線形探索 ～ 58
2－3　基本アルゴリズム(探索)　その2　～ 2分探索 ～ 63
2－4　基本アルゴリズム(整列)　その1　～ 選択法 ～ 70
2－5　基本アルゴリズム(整列)　その2　～ 交換法 ～ 80
2－6　基本アルゴリズム(整列)　その3　～ 挿入法 ～ 88
2－7　再帰 96
2－8　高速な整列アルゴリズム　～ クイックソート ～ 98
2－9　その他の整列アルゴリズム 104
2－10　文字列操作アルゴリズム　その1　～ 文字列の照合 ～ 107
2－11　文字列操作アルゴリズム　その2　～ 文字列の置換 ～ 113
2－12　文字列操作アルゴリズム　その3　～ 文字列の圧縮 ～ 117

Part3　データ構造 125
3－1　データ構造の基礎知識 126
3－2　リスト 130
3－3　スタック 141
3－4　キュー 145
3－5　ハッシュ表 152
3－6　木 158
3－7　2分探索木 162

3－8　ヒープ……166
3－9　木の巡回……174
3－10　B木……179
3－11　グラフ……182
3－12　最短経路探索……184

Part4　オブジェクト指向……189
4－1　オブジェクト指向の基礎知識……190
4－2　オブジェクト指向を活用したプログラム……196

Part5　応用アルゴリズム……203
5－1　ファイル処理……204
5－2　ファイルの併合……207
5－3　ファイルの突合せ……210
5－4　コントロールブレイク処理……214

Part6　アルゴリズムパターン集……217
パターン1　連続範囲の処理……218
パターン2　構造体配列……221
パターン3　順位づけ……224
パターン4　配列で数値表現……228
パターン5　文字列の区切り……234
パターン6　ポインタ配列……237
パターン7　画像データ処理……239
パターン8　ビットデータ(2進数)の操作……243

付録

擬似言語の記述形式……249

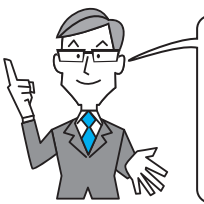
索引……253

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(日本産業規格)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。

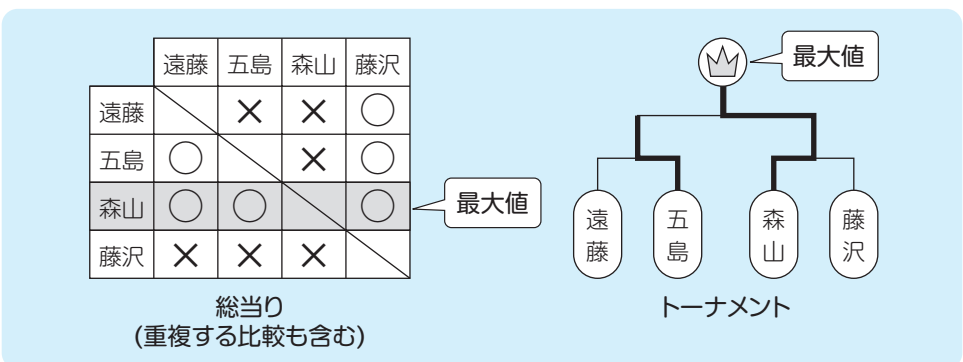
理解を深めよう

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，
4×3＝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。

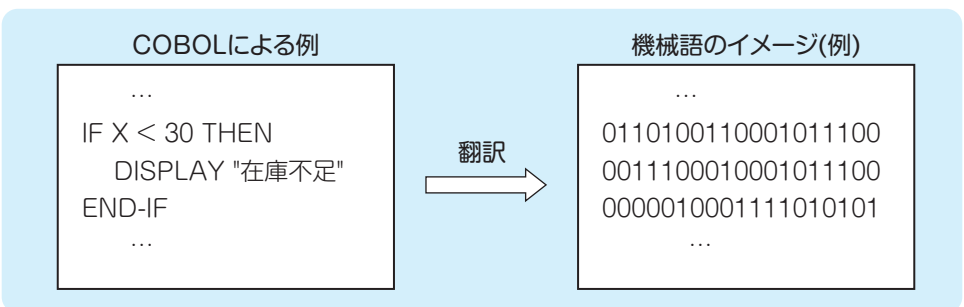
コンピュータのCPUは，こうした比較処理を一つひとつ処理していきますから，同じ結果が得られるのであれば，より少ない処理となるようなアルゴリズムのほうがよいことがわかるでしょう。

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。

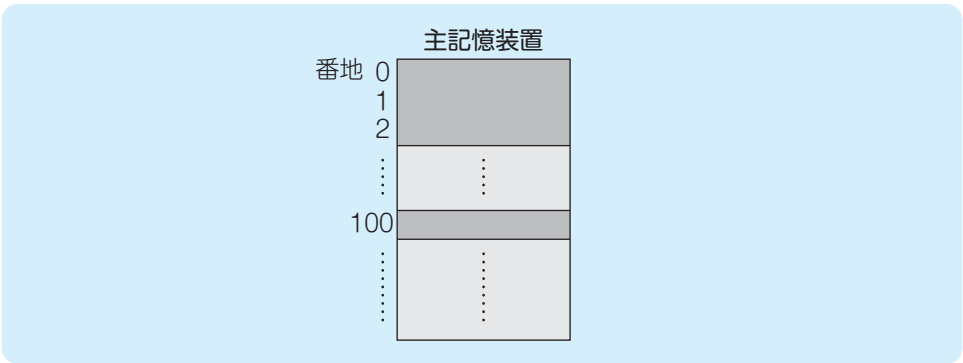
たとえば，上記の図のCOBOLによる記述は，「Xが30 より小さかったら“在庫不足”と表示する」ことを示しています。在庫数が30未満であれば，在庫不足として注意を促すような目的をもつアルゴリズムをCOBOLで表現したものといえます。

データの格納場所と記憶装置

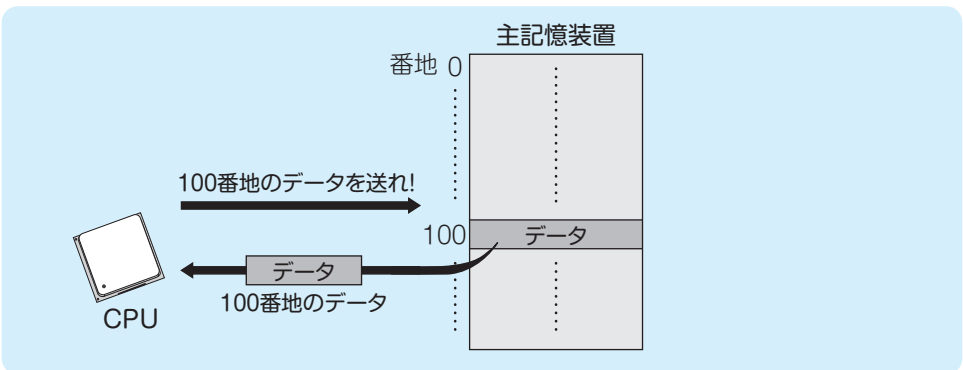
コンピュータに何か作業をさせる場合には，必要なデータを与えておかなければなりません。そのためにはデータを格納する場所がなければなりません。また，処理した後のデータを格納する場所も必要となります。この役割を担うのが記憶装置です。

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Java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”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

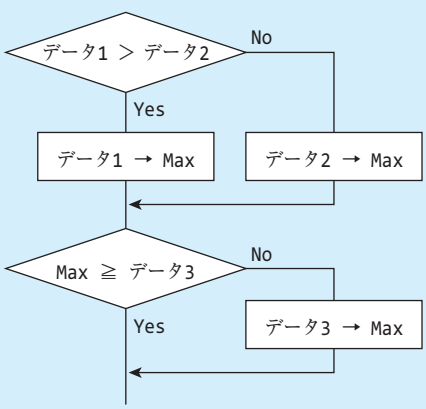
ここでは，アルゴリズムの基礎知識として，変数や定数について学習します。

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代表例に，**流れ図**（**フローチャート**）と**擬似言語**があります。

流れ図(フローチャート)



擬似言語

```
if (データ1 が データ2 より大きい)
  Max ← データ1
else
  Max ← データ2
endif
if (データ3 が Max より大きい)
  Max ← データ3
endif
```

流れ図には視覚的なわかり易さがあります。その反面，記述の自由度が高く，注意しないと煩雑で分かりにくいものになりがちです。擬似言語は視覚的なわかりやすさこそ流れ図に譲りますが，プログラム言語に近くプログラミングとの相性は抜群です。

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。

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。

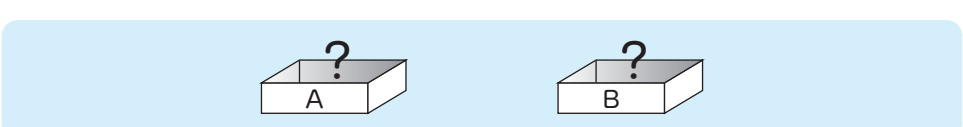
①変数には名前（変数名）がついている

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。この場合，変数に値が格納されていない状態として，**未定義**（不定）とよびます。つまり，変数Aを用意しただけでは，その変数Aは最初は“未定義”となります。

この性質は重要です。



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，擬似言語において，変数Aに10（数値定数）を代入するときには，

A ← 10　（あるいは　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

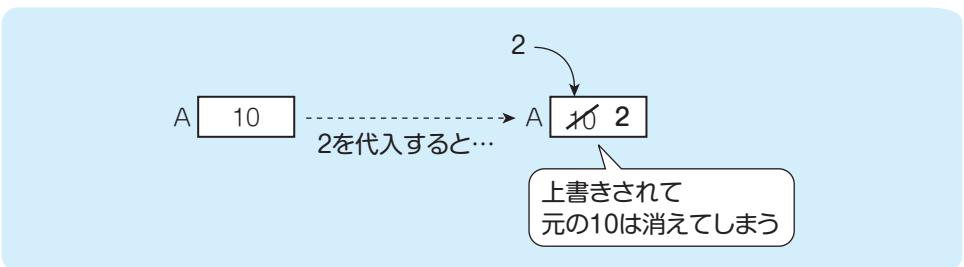
納するための変数に文字を代入したり，文字を格納するための変数に数値を代入することはできません。なお，変数に**初期値**（最初の値）を代入することを，**初期化**といいます。

また，変数に**未定義の値**を代入することもでき，その場合，その変数は“未定義”になります。

A ← 未定義の値

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。

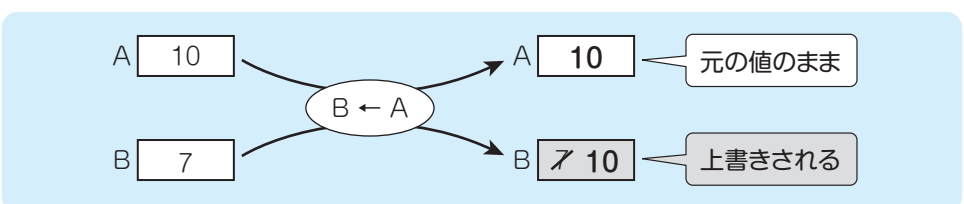


⑤変数には，ほかの変数の値を代入することができる

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，

B ← A

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

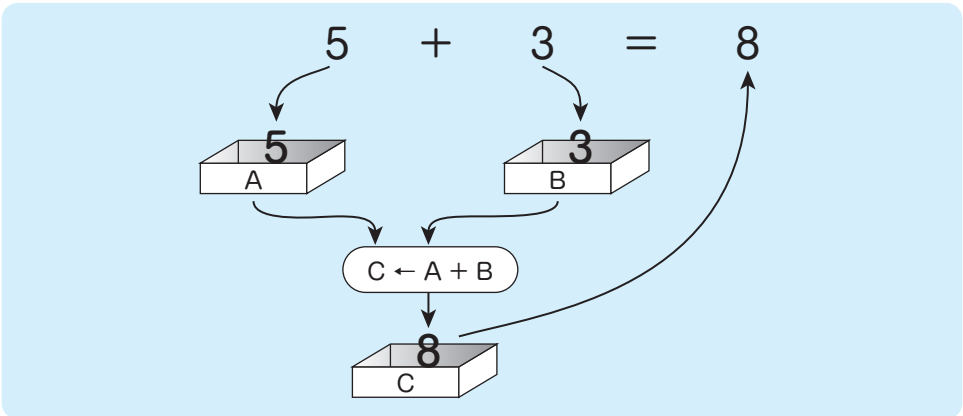
変数には，演算の結果を代入することができます。たとえば，

B ← A ＋ 5

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，

C ← A ＋ B

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，「C ← A ＋ B」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。



演算子には，通常の四則演算子（＋－×÷）のほかに，剰余を求める**mod**を使います。以下は，式の一例です。

式	意　味
B ← A × 2	Bには「Aを2倍した値」を代入する
C ← A ＋ B	Cには「AとBの和」を代入する
B ← A mod 2	Bには「Aを2で割った余り」を代入する
A ← B ＋ C ÷ D	Aには「B」＋「CをDで割った値」を代入する
A ← (B ＋ C) ÷ D	Aには「BとCの和をDで割った値」を代入する

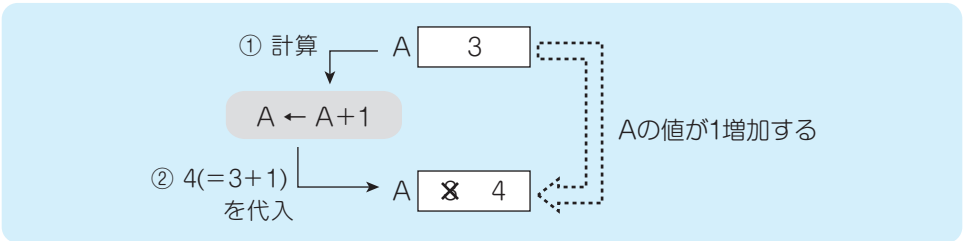
変数の内容は，代入によって変わるものであり，式によって変わるものではないことに注意してください。たとえば，「B ← A × 2」を実行したとき，Bの値はA×2で更新されますが，Aの内容は変化しません。

⑦自分自身に計算結果を代入することができる

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。これは，ある変数の値を増加／減少させる演算です。以下に一例を示します。

式	意　味
A ← A ＋ 1	変数Aを1増加させる
A ← A － B	変数AからBの値を減じる
A ← A × 2	変数Aを2倍する
A ← A ÷ B	変数Aの値を1/B倍する

たとえば，変数Aの内容が3であるときに「A ← A ＋ 1」が実行されたときには，次の順序で計算・代入が行われ，結果としてAが1増加したことになるのです。



変数の値を増減する処理は，**インクリメント**（増加）／**デクリメント**（減少）ともよばれ，プログラムでは頻出します。

インクリメントやデクリメントを行う変数は，適切に初期化されていなければなりません。なぜなら，変数の値が未定義であった場合，それを増加・減少しても意味はないからです。

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"，'A'

というように，ダブルクォーテーション（" "）やシング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"，'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，

A ← 5 + 3

と記述します。"5"，'3'のようにダブルクォーテーションやシング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。

理解を深めよう

型と宣言

変数は，取り扱うデータの種類によって，いくつかのグループに分けられます。このグループを**型**とよびます。代表的な型には次のものがあります。

整数型	整数を取り扱う変数
実数型	実数を取り扱う変数
文字型	文字を取り扱う変数
文字列型	文字のまとまり(文字列)を取り扱う変数
論理型	true(真)またはfalse(偽)の2値を取り扱う変数

整数型の変数は，実数を保持できません。ある式の値を整数型変数に格納するような場合には，値の小数点以下が切り捨てられて代入されます。

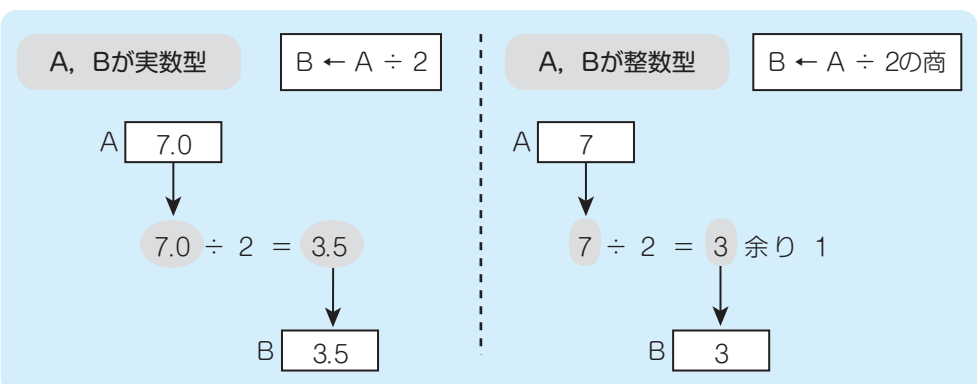
また，擬似言語では，「整数同士の除算で，整数の商を結果として得る」場合，

　　7 ÷ 2 の商

のように記述することで，

　　7 ÷ 2 ＝ 3 余り 1　　（7＝3×2＋1の3が商，1が剰余）

という除算結果の商である 3 が求められます。



擬似言語では，変数を用いる場合に前もって変数を宣言します。変数宣言は，型と変数名との組合せで行います。

```
整数型: N　　← 整数型変数 N を用いる
実数型: R　　← 実数型変数 R を用いる
文字型: C　　← 文字型変数 C を用いる
文字列型: S　← 文字列型変数 S を用いる
論理型: F　　← 論理型変数 F を用いる
```

変数の型と定数の代入

整数や実数の定数は，普段用いている書き方で指定できます。整数値であっても，実数として取り扱っていることを明示するために「.0」をつけることもあります。

```
整数型: N
実数型: R
N ← 3
R ← 3.0
```

文字定数は，変数と区別するためにダブルクォーテーションやシングルクォーテーションで囲みます。

```
文字型: C
文字列型: Word
C ← "A"          ← 文字型変数Cに，文字Aを代入
Word ← "ABC"     ← 文字列型変数Wordに，文字列ABCを代入
```

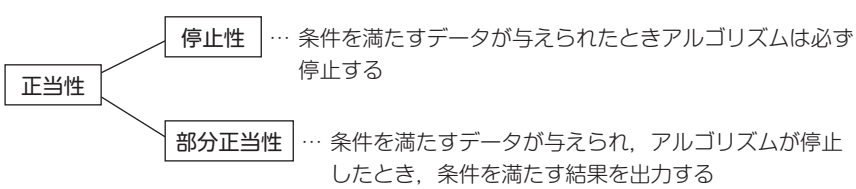
論理型変数には，trueまたはfalseを設定します。

```
論理型: Flag
Flag ← true
```

参考：アルゴリズムの正当性

アルゴリズムは正しくなければなりません。アルゴリズムが正しいことを評価する基準に正当性があります。

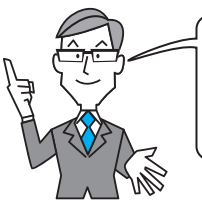
正当性は停止性と部分正当性という側面をもちます。



停止性と部分正当性をあわせた基準を，全正当性とよびます。全正当性を満たすアルゴリズムは，条件を満たすデータが与えられたとき「必ず停止し，かつ出力条件を満たす結果を出力」します。

1-3　基本制御構造　その1

～ 順次と分岐 ～



アルゴリズムの表現方法である擬似言語と流れ図(フローチャート)を用いて，基本制御構造とよばれる三つの処理構造を学習します。ここでは，順次と分岐を取り上げます。

基本知識の整理

基本制御構造

プログラム構造の基本となるのは，

順次

選択（分岐）

繰返し（ループ）

の三つの制御構造であり，これらを**基本制御構造**といいます。

また，この基本制御構造だけを用いて読み易いプログラムを書く，という考え方を**構造化プログラミング**といいます。

擬似言語プログラムの書き方

擬似言語プログラムは，**宣言部**と**処理部**から構成されます。

プログラム	宣言部	プログラム(手続，関数)や変数を宣言する
	処理部	プログラムで行う処理を記述する

変数の宣言では，プログラムで使用する変数について，型と変数のリストを記述します。同じ型の変数であれば，1行にまとめても構いません。また，宣言と同時に，初期値を代入することもできます。

```
整数型: A, B, C          /* 三つの整数型変数A，B，Cを用いる */
実数型: Rate ← 2.5       /* 初期値を2.5とした実数型変数Rateを用いる */
```

処理部には，プログラムで実行する命令を順番に記述します。

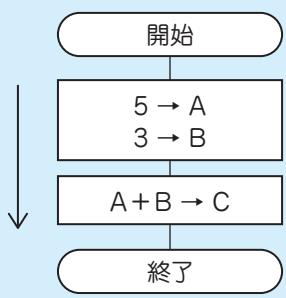
```
A ← 0        /* 最初の処理：Aに0を代入する */
B ← A        /* 次の処理 */
C ← B        /* 最後の処理：これでA～Cがすべて0になった */
```

記述された各処理は，原則として上から下へ順に処理されていきます。このような制御構造を，“**順次**”といいます。

流れ図の書き方

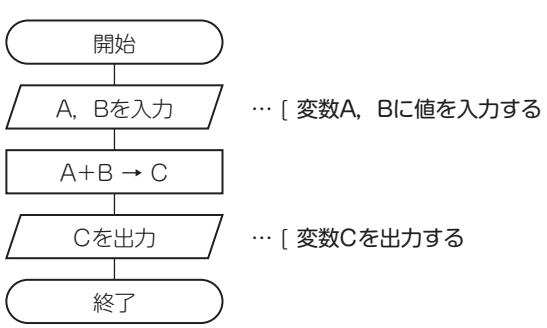
流れ図は，一番上の開始を表す端子から処理が始まり，一番下の終了を表す端子で処理が終わります。実行する処理は，長方形の記号（基本処理記号）の中に記述します。

流れ図に記述された処理は，擬似言語と同様に上から下へ順に処理されていく（順次）と考えてください。一つの長方形の中に，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ます。



参考：データの入出力(流れ図)

流れ図では，データの入出力を記述するための記号があります。平行四辺形の記号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。



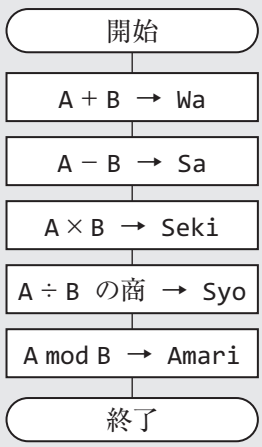
Example

変数A，B（共に整数型，値は格納済み）の四則演算を行うアルゴリズムを，擬似言語及び流れ図で表したものです。A，Bの和（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，割り算の余りをAmariという変数にそれぞれ格納します。

〔擬似言語〕

```
○Calculate()
  整数型: A, B, Wa, Sa, Seki, Syo, Amari
  Wa ← A + B
  Sa ← A − B
  Seki ← A × B
  Syo ← A ÷ B の商
  Amari ← A mod B
```

〔流れ図〕

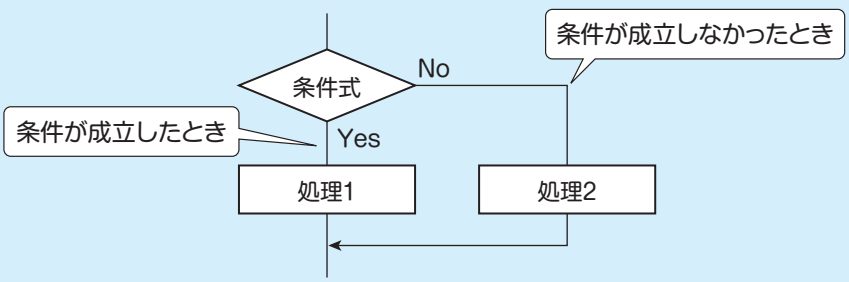


分岐(選択)

条件によって，実行する処理を分けることを，“**分岐**(**選択**)”といいます。
擬似言語プログラムでは，**if**文(**if ～ else**文)を用いて，次のように分岐を記述します。

```
if (条件式)
  処理1        … 条件が成立したときの処理
else
  処理2        … 条件が成立しなかったときの処理
endif
```

一方，流れ図では，ひし形の記号に条件式を記述し，その条件の判定結果によって，次のようにYes，Noに分岐させます。



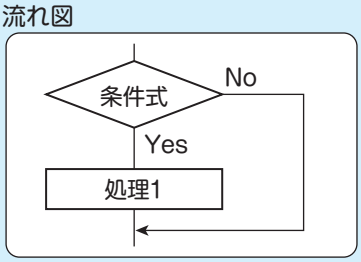
これらの例では，条件が成立した場合(条件式が真の場合)には，処理1が実行されます。このときには処理2は実行されないわけです。逆に，条件が成立しなかった場合(条件式が偽の場合)には，処理2が実行され，処理1は実行されません。

また，条件が成立しなかった場合に何も処理を行わないのであれば，次のように記述します。

擬似言語

```
if (条件式)
  処理1
endif
```

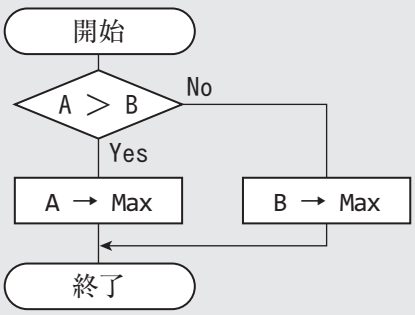
流れ図



Example

変数A，B(共に整数型，値は格納済み)のうち，大きい方の値を変数Maxに格納します。

```
if (A が B より大きい)
  Max ← A
else
  Max ← B
endif
```

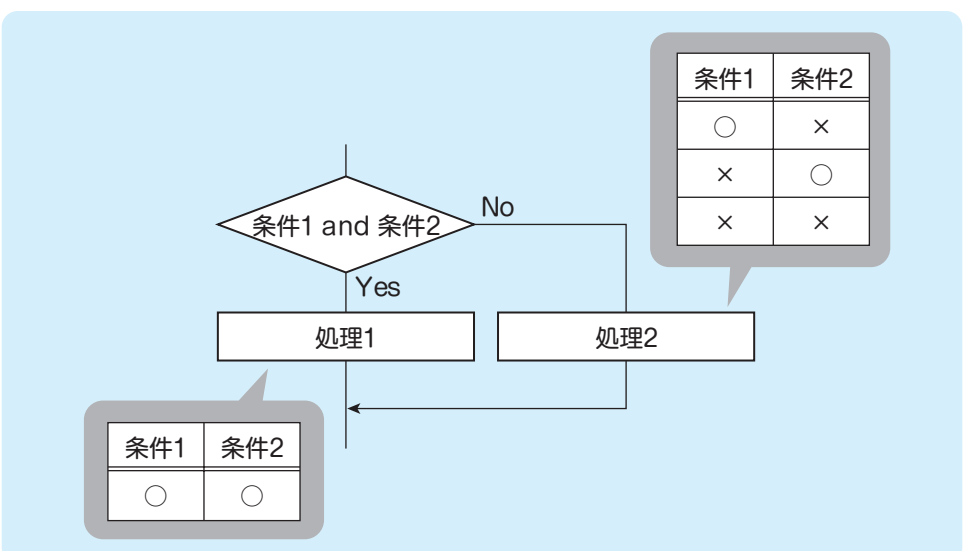


理解を深めよう

複合条件

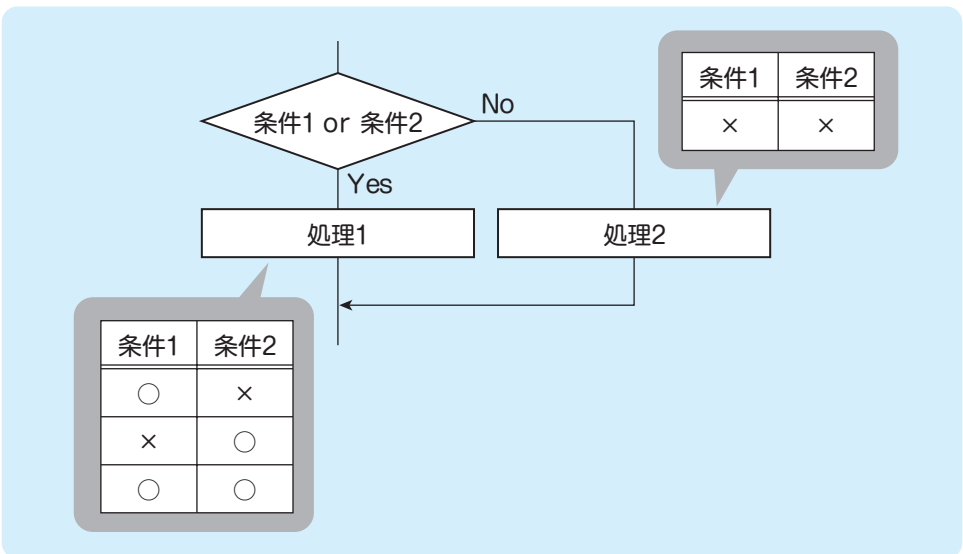
複数の条件を“and(かつ)”や“or(または)”で結んだものを**複合条件**とよびます。この“**and**”は次の図のように条件1と条件2がともに満たされたときYes側に進み，どちらか一方の条件が満たされない場合や，ともに満たされない場合はNo側へ進むことになります。

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 and 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。



“and”で結ばれた条件

次に“or(または)”を考えてみましょう。この“**or**”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 or 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。



“or”で結ばれた条件

また，条件を入れ子構造にすることもできます。

Example

(1) 年齢(Age)が20代であれば顧客区分(Div)を“A”に，そうでなければ“B”にする。

〔プログラム1〕

```
if ((Age が 20 以上) and (Age が 29 以下))
  Div ← "A"
else
  Div ← "B"
endif
```

〔プログラム2〕

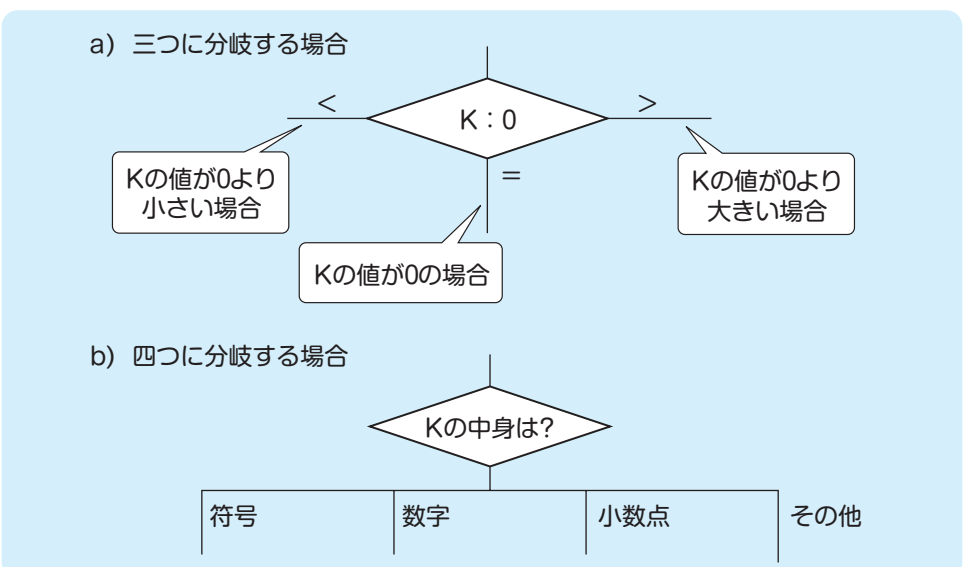
```
Div ← "B"
if ((Age が 20 以上) and (Age が 29 以下))
  Div ← "A"
endif
```

(2)　年齢(Age)が20代の男性(S＝1)の顧客区分(Div)を“A”に，20代女性(S＝2)の顧客区分を“B”に，それ以外を“C”にする。

```
Div ← "C"
if ((Age が 20 以上) and (Age が 29 以下))
  if (S が 1 と等しい)
    Div ← "A"
  else
    Div ← "B"
  endif
endif
```

多分岐

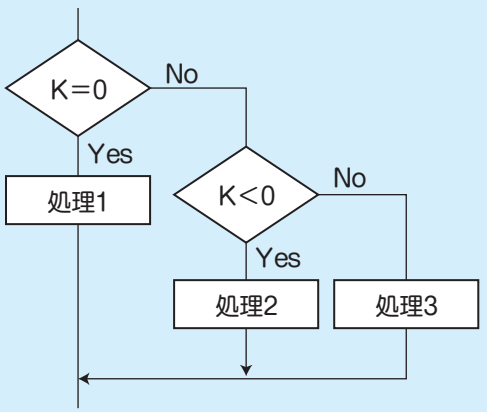
ここまで説明してきた分岐は，処理が二つに分かれる構造ですから“**二分岐**”とよびます。これに対して三つ以上に分岐先が分かれる分岐の構造を“**多分岐**”とよびます。流れ図では，多分岐を次のように記述できます。



多分岐の構造

擬似言語プログラムでは，if文に「**elseif**」を組み込んで，次のように記述します。

```
if (K が 0 と等しい)
  処理1 … K=0のときの処理
elseif (K が 0 より小さい)
  処理2 … K<0のときの処理
else
  処理3 … K>0のときの処理
endif
```



このif文では，条件式を上から順に評価し，最初に真になった条件式のところに記述された処理を実行します。条件式のいずれも真にならなかった場合は，elseに記述された処理を実行します。

なお，elseifは複数組み込むことができます。また，条件式のいずれも真にならなかった場合のelseを省略することもできます。