《パターン②》

次の例を見ると、代入は置換えであることがわかります。代入を実行すると、変数xの元の値はなかったことにされ、新しい値に上書きされます。また、変数yには元の値がそのまま残されることから、変数yからxへの値のコピーともいえます。

②の例　変数 y を変数 x に代入する

x ← y

《実行前》 x 3　y 5

《実行後》 x 5　y 5

《パターン③》

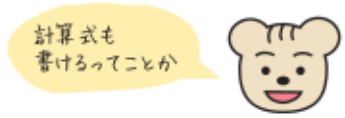
今度は、変数yの値を使って演算を行い、その結果を変数xに代入しています。計算は、単純な四則演算のほか、割り算の余りを求める剰余算なども使えます。

③の例　y に 4 を掛けたものを変数 x に代入する

x ← y × 4

《実行前》 x 5　y 5

《実行後》 x 20　y 5



《パターン④》

アルゴリズムやプログラムでよく見かけるパターンです。変数xが←の左と右の2か所に出てきて混乱するかもしれませんが、アルゴリズムやプログラムでは、あたりまえのように登場する書き方なので、慣れておきましょう。

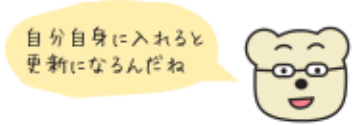
命令の意味は、矢印の右側の変数xの値（元の値）を使って計算を行い、その結果を変数xの新しい値として置き換えるということです。

④の例　変数 x の値に、5 を掛けたものを元の変数 x に代入する

x ← x × 5

《実行前》 x 20

《実行後》 x 100



代入は、アルゴリズムを書くうえで、最もよく使うものなので、少し練習しておきましょう。次の問題にチャレンジしてみてください。

問題にチャレンジ！

次の1～8の命令をすべて実行したとき、変数a、b、cの値を求めよ。なお、左端の数字は、説明のために付加した行番号である。

```
1　a ← 1
2　b ← 2
3　c ← 5
4　a ← a + 1
5　b ← a + b × c
6　c ← c + 2c + 3c
7　b ← a × (b + b)
8　a ← b − c − a
```

解き方　各行を実行した後の変数の値は、1～3行目：a＝1、b＝2、c＝5、4行目：a＝2、b＝2、c＝5、5行目：a＝2、b＝12、c＝5、6行目：a＝2、b＝12、c＝30、7行目：a＝2、b＝48、c＝30、8行目：a＝16、b＝48、c＝30　となります。

●“選択”は、実行する処理を振り分けること

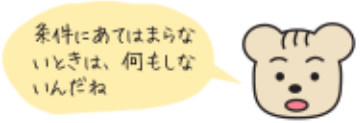
もう一つのベーシックな命令である**選択**は、条件によって値を判断した結果で、その後に行う処理を振り分ける機能をもちます。

例えば、「もし変数xの値が1だったら、変数yを10とせよ」という命令であれば、判断の結果として行われる処理は、次のようになります。

《実行前》 x 1　y 5

《実行後》 x 1　y 10　（変数xが1なので、yに10が代入される）

上記の例は、「変数xの値が1でなかったとき」、つまり「条件にあてはまらなかったとき」は、後の処理（yに10を代入）は行いません。



今度は、「条件にあてはまるとき」と「あてはまらないとき」で、処理の方法が二つに分かれる場合を見てみましょう。「もし変数xの値が2だったら変数yを100とし、変数x

繰返し処理とその構造

繰返し処理は選択処理と順次処理を組み合わせたもので、変数の値を変化させながら条件文に合致しなくなるまで処理を繰り返します。「1〜1000までの値を加算」、「500の要素を並べ換える」といった計算を人が手作業でやるのはたいへんでミスも起こりやすいものです。繰返し処理のプログラムを使い、コンピュータで処理すれば一瞬で作業が完了します。つまり、コンピュータの真骨頂ともいえる作業が繰返し処理です。

●繰返し処理に必要なもの

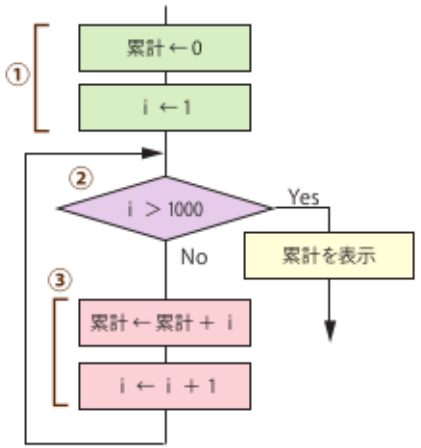
繰返し処理には、**①変数の初期値設定**、**②繰返し終了条件の判定**、**③累計処理および制御変数の更新**の設定が必要です。例えば1〜1000まで累計するなら、まず累計変数を0に設定したうえで、制御変数に初期値1を設定（①）し、1000を超えるまで（②）、1ずつ更新（③）しながら、累計変数に加えていきます。これら三つの処理を組み合わせることで、繰返し処理が成り立ちます。

《繰返し処理のポイント》

① 変数の初期値の設定
- 累計値を入れておく変数「累計」を0にしておく
- 加える数（変数ｉ）の初期値を1にする

② 繰返し終了条件の判定
- 加える数が1000を超えているか？
（超えていたら、繰返し処理を終了）

③ 累計処理および制御変数の更新
- 変数「累計」に加える数（i）を加算する
- 加える数（i）を1増加させ、累計処理を繰り返す



図表2-3-5　繰返しの流れ図

この例では、変数ｉは「加える数」としての役割のほかに、繰返しの条件にあてはまるかどうかを判断するための値としても使われています。このように、繰返しの制御に使う変数のことを、**制御変数**と呼びます。

なお、この例の変数ｉのように、値の更新をするために繰り返すごとに値を1ずつ増やしていく処理のことを**インクリメント**（増分）、反対に1ずつ減らしていくことを**デクリメント**（減分）と呼ぶことがあります。

擬似言語の繰返し判断は、必ず「継続条件」にする

繰返しの条件文を書くときに迷いやすいのが、「**終了条件**」か「**継続条件**」なのかということ。p.36でも取り上げましたが、もう少し具体的に解説します。

左ページの例で、「1000を超えたらループ処理を抜けて終了する」というのは**終了条件**で、**継続条件**では「1000を超えなければループ処理を継続する」になります。言葉で書けば、どちらも同じ意味ですが、単に「ｉ＞1000」のように式で記述されていると、終了の条件か継続の条件なのか、判断ができません。

擬似言語の仕様では、このような条件記述については、「条件が真の間、処理を繰返し実行する」、つまり必ず継続条件で記述すると決められています。

●繰返し条件が終了条件のときは反転が必要

繰返し条件が、例えば「変数ｉの値が1000を超えたら（1000より大きい）繰返し処理を終了する」という終了条件になっていたら、擬似言語で条件式を記述するときは「変数ｉの値が1000以下なら繰返しを続ける」という継続条件に反転します。

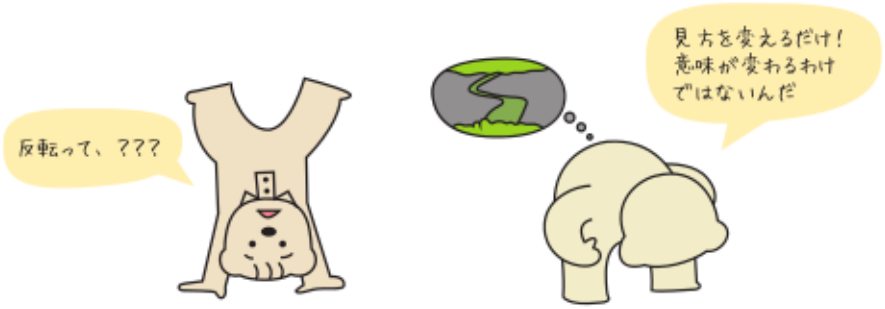
このとき、特に関係演算子の反転には注意が必要です。比較演算子を使った条件式を反転すると、それぞれ、「a＞bは、a≦b」、「a＜bは、a≧b」、「a≧bは、a＜b」、「a≦bは、a＞b」となります。「以上」と「より大きい」、「未満」と「以下」では、含まれる値の範囲が違うことを意識しておきましょう。

《繰返し条件の反転とは……》

終了条件で書かれていたら…　**if（i ＞ 1000）**　…変数ｉが1000より大きくなったら終了

条件を反転する

擬似言語では継続条件に変換する　**if（i ≦ 1000）**　…変数ｉが1000以下なら繰返しを継続



テーマ 3-4

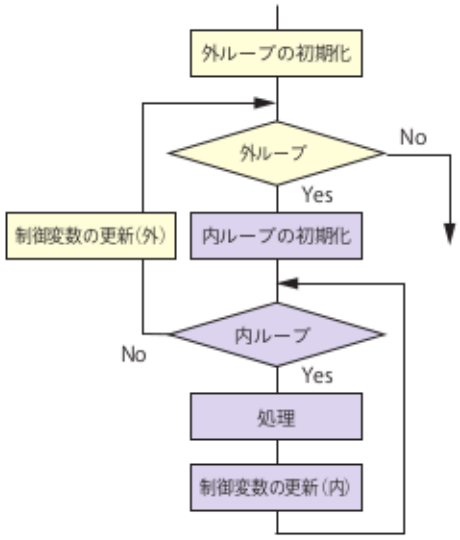
多重の繰返し処理 — for文 —

多重の繰返しとは、繰返し処理の中に繰返し処理が含まれる、いわば入れ子の構造です。複雑なプログラムになりますが、ループ端記号による流れ図で記述すると、入れ子の構造が視認しやすくなります。

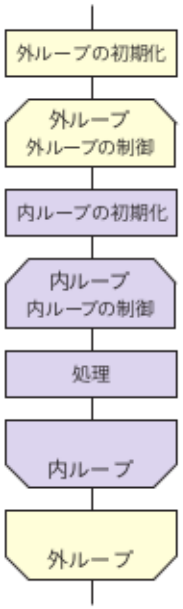
多重の繰返し処理では、繰返し部分が入れ子になっており、複数の制御変数を同時に使って繰返し処理を制御します。代表的な用途としては、二次元配列を使った処理が挙げられます。二次元配列を扱うプログラムでは、処理の対象となる要素の位置（要素番号）を変化させるために、要素の行番号と列番号を別々に制御します。また、二つの一次元配列を個別に制御する文字列照合などでも、多重の繰返しを使います。

なお、多重の繰返し処理の流れ図は、下図のようにループ端記号を使ったほうがスッキリ記述でき、処理がわかりやすくなります。

〔判断記号による多重繰返し〕



〔ループ端記号による多重繰返し〕



図表3-4-1　多重繰返し処理の流れ図

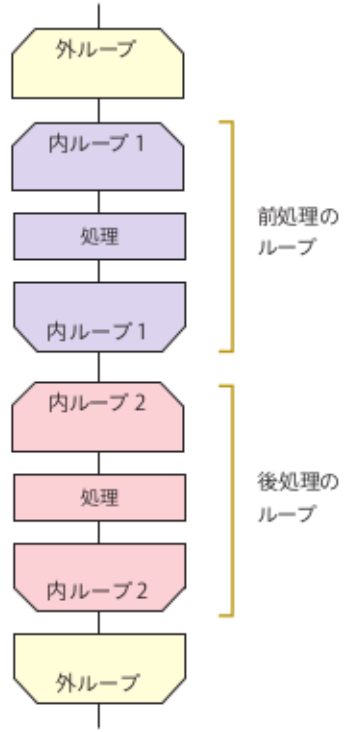
●流れ図の概要

多重繰返しで注目したいのは、外ループにも内ループにも、「初期化（ループに入る前の初期設定）」、「処理」、「制御変数の更新」の3要件が必要になること。

前ページの図表3-4-1で、各要件がどの位置にあるべきかを把握しておくと、抜け落ちが見つけやすくなります。なお、ループ端記号の流れ図では、「制御変数の更新」を「ループ開始端（または終端）の制御記述」の中で行います。

また、右のように繰返しの中に、二つの繰返しが入っており、順に処理を進める入れ子構造もあります。こちらは、例えば二次元配列の操作において、前処理と後処理のように役割を分けるときに用います。

〔二つの内ループを順に行う形〕



図表3-4-2　多重繰返しの形態

多重の繰返し処理を考えてみよう

代表的な二次元配列の操作として、掛け算の九九表の作成があります。九九表は、1×1、1×2、…、1×9と1の段を計算したら、2の段2×1、2×2、……と段を移動しながら積を求めていきます。これを多重の繰返し処理を使って考えていきましょう。

例題

「九九表を出力するアルゴリズム」

九九の1段分の値を1行として、1～9の段までの印刷を行う。

・値の印刷は、関数“印刷（atai）”により、値と文字の区切りとなるスペースを印刷できるものとする。

・1段分の印刷が終わったら、1行分の改行を行う。この処理は、関数“印刷(改行)”によって行う。

トレース問題を解いてみよう

実際の試験問題を解いてみましょう。この問題は、引数として二次元配列を受け取り、配列要素を参照しながら、別の二次元配列に入れて返すというものです。

例題

「行列データの変換」

次の記述中の［ a ］～［ c ］に入れる正しい答えの組合せを、解答群の中から選べ。ここで，配列の要素番号は1から始まる。

要素の多くが0の行列を疎行列という。次のプログラムは、二次元配列に格納された行列のデータ量を削減するために、疎行列の格納に適したデータ構造に変換する。

関数transformSparseMatrixは、引数 matrix で二次元配列として与えられた行列を、整数型配列の配列に変換して返す。関数 transformSparseMatrix を transformSparseMatrix({{3, 0, 0, 0, 0}, {0, 2, 2, 0, 0}, {0, 0, 0, 1, 3}, {0, 0, 0, 2, 0}, {0, 0, 0, 0, 1}})として呼び出したときの戻り値は、{{［ a ］},{［ b ］},{［ c ］}}である。

〔プログラム〕

```
○整数型配列の配列: transformSparseMatrix(整数型の二次元配列: matrix)
  整数型: i, j
  整数型配列の配列: sparseMatrix
  sparseMatrix ← {{}, {}, {}} /* 要素数0の配列を三つ要素にもつ配列 */
  for (i を 1 から matrixの行数 まで 1 ずつ増やす)
    for (j を 1 から matrixの列数 まで 1 ずつ増やす)
      if (matrix[i, j] が 0 でない)
        sparseMatrix[1]の末尾 に iの値 を追加する
        sparseMatrix[2]の末尾 に jの値 を追加する
        sparseMatrix[3]の末尾 に matrix[i, j]の値 を追加する
      endif
    endfor
  endfor
  return sparseMatrix
```


解答群

	a	b	c
ア	1, 2, 2, 3, 3, 4, 5	1, 2, 3, 4, 5, 4, 5	3, 2, 2, 1, 2, 3, 1
イ	1, 2, 2, 3, 3, 4, 5	1, 2, 3, 4, 5, 4, 5	3, 2, 2, 1, 3, 2, 1
ウ	1, 2, 3, 4, 5, 4, 5	1, 2, 2, 3, 3, 4, 5	3, 2, 2, 1, 2, 3, 1
エ	1, 2, 3, 4, 5, 4, 5	1, 2, 2, 3, 3, 4, 5	3, 2, 2, 1, 3, 2, 1

出典：2022年4月公開 基本情報技術者試験 科目B サンプル問題 問4

プログラム冒頭で呼び出している関数の、（ ）内の値が引数の配列に代入されるデータで、図のような5行5列の配列になります。これを使ってトレースしてみましょう。

引数の配列：matrix

列(j) / 行(i)	[1]	[2]	[3]	[4]	[5]
[1]	3	0	0	0	0
[2]	0	2	2	0	0
[3]	0	0	0	1	3
[4]	0	0	0	2	0
[5]	0	0	0	0	1

処理結果

戻り値の配列：sparseMatrix

[1] 解答群のa	[2] 解答群のb	[3] 解答群のc

図表4-1-1　処理の概要

for文の繰返しは2重ループになっており、要素番号をi（行番号）とj（列番号）として、1行1列目→1行2列目…2行1列目→2行2列目…の順に1行1列～5行5列まで、下記の処理を行っていきます。

```
      if (matrix[i, j] が 0 でない)
        sparseMatrix[1]の末尾 に iの値 を追加する
        sparseMatrix[2]の末尾 に jの値 を追加する
        sparseMatrix[3]の末尾 に matrix[i, j]の値 を追加する
      endif
```

順にトレースしてみましょう。matrix[i, j] が0でない要素（7つある）を対象として、次の処理を行っていけばよいのです。プログラム中に言葉で書かれている処理「末尾に追加する」は、選択肢を見ると「,」で区切って値を追加すればよいことがわかります。

1行1列　値［ 3 ］

sparseMatrix[1] →［ 1 ］（行番号）

sparseMatrix[2] →［ 1 ］（列番号）

sparseMatrix[3] →［ 3 ］（値）

テーマ 6-3

複数のオブジェクトを関連付ける

ここまでのオブジェクト指向プログラミングは、データ構造を生成したり、要素を追加したりというものでしたが、オブジェクト指向の本質は中の構造を意識せずに使用できること。そんな使い方を見ていきまょう。

オブジェクト指向プログラミングの利点は、必要な機能を呼んでくればすぐに使えることです。また、必要に応じた機能を新たなクラスとして定義すれば、それらを組み合わせることもできます。ここでは、複数のオブジェクトを組み合わせて一つの機能を実現するプログラミングの例を取り上げます。アルゴリズムとしての複雑さはありませんが、実際のプログラミングではよく使うパターンです。しっかり理解しておきましょう。

オブジェクト間の関係

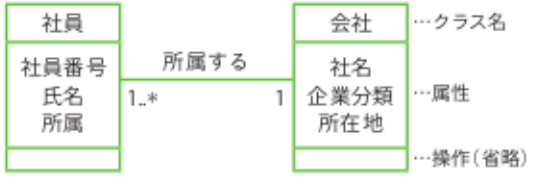
オブジェクトのクラス間には、さまざまな関係があります。代表的なものを挙げます。

関連：「あるクラスと別のクラスに何らかのつながりがある」ことを**関連**と呼びます。

is-a 関係：「あるクラス（**サブクラス**）が他のクラス（**スーパークラス**）の一種である」という関係。スーパークラスからサブクラスへ展開することを**特化**、サブクラスからスーパークラスを作成することを**汎化**と呼びます。

part-of 関係：「あるクラスが他のクラスの一部である」という関係。つまり、あるオブジェクトが複数のオブジェクトによって構成されているということになります。全体を部分を表すクラスへ展開することを**分解**、部分を表すクラスを一つのクラスにまとめることを**集約**と呼びます。

また、それぞれの関係は、UMLのクラス図を使って表すのが一般的です。例として、「会社」と「社員」の関連を表現してみましょう。なお、線の下の記号は**多重度**（右表）です。



図表6-3-1　クラス図と多重度の表記

《多重度の表記》

記号	説明
1	1
*	0以上の多数
0..1	0または1
0..*	0以上
1..*	1以上

左ページのクラス図では、「会社」クラスと「社員」クラスに必要な属性を記入しています。読み方は「社員は会社に所属する」であり、多重度は「一つの会社に1人以上の社員がいることを示しています。

複数オブジェクト間の「関連」を取り上げた問題

次は、二つのクラスによるショッピングカートをテーマにした問題です。ショッピングカートを表すオブジェクトと、本を表すオブジェクトがあり両者には「関連」の関係があります。また、ショッピングカートの中に複数の本が含まれる（参照している）ことから、「集約」の関係があるともいえます。UMLで記述する場合は記号が異なりますが、オブジェクトの性質や見方によって、どちらで記述してもかまいません。

例題

「ショッピングカートの合計額出力」

次のプログラム中の［ a ］と［ b ］に入れる正しい答えの組合せを、解答群の中から選べ。

書籍の情報を表すクラスBookと、ショッピングカートを表すクラスCartを利用したプログラムである。プログラムでは、ショッピングカートに複数の書籍を追加して、カート内の書籍の情報と合計金額を出力する処理を行う。

コンストラクタ	説明
Book(文字列型:title, 文字列型:author, 整数型:price)	引数title、author、priceの値で、メンバ変数title、author、priceをそれぞれ初期化する。

メンバ変数	型	説明
title	文字列型	書籍のタイトル。
author	文字列型	書籍の著者。
price	整数型	書籍の価格。

メソッド	戻り値	説明
getInfo()	文字列型	インスタンスの内容の文字列表現（"書籍[タイトル, 著者, 価格]"の形式）を返す。

表 クラスBookの説明

第6章　演習問題

問　次のプログラム中の［　　］に入れる正しい答えを、解答群の中から選べ。

手続delNodeは、単方向リストから、引数posで指定された位置の要素を削除する手続である。引数posは、リストの要素数以下の正の整数とする。リストの先頭の位置を1とする。

クラスListElementは、単方向リストの要素を表す。クラスListElementのメンバ変数の説明を表に示す。ListElement型の変数はクラスListElementのインスタンスの参照を格納するものとする。大域変数listHeadには、リストの先頭要素の参照があらかじめ格納されている。

表　クラスListElementのメンバ変数の説明

メンバ変数	型	説明
val	文字型	要素の値
next	ListElement	次の要素の参照 次の要素がないときの状態は未定義

〔プログラム〕

```
大域: ListElement: listHead  /* リストの先頭要素が格納されている */

○delNode(整数型: pos)  /* posは、リストの要素数以下の正の整数 */
  ListElement: prev
  整数型: i
  if (pos が 1 と等しい)
    listHead ← listHead.next
  else
    prev ← listHead
    /* posが2と等しいときは繰返し処理を実行しない */
    for (i を 2 から pos − 1 まで 1 ずつ増やす)
      prev ← prev.next
    endfor
    prev.next ← [　　]
  endif
```


解答群

ア　listHead　　イ　listHead.next

ウ　listHead.next.next　　エ　prev

オ　prev.next　　カ　prev.next.next

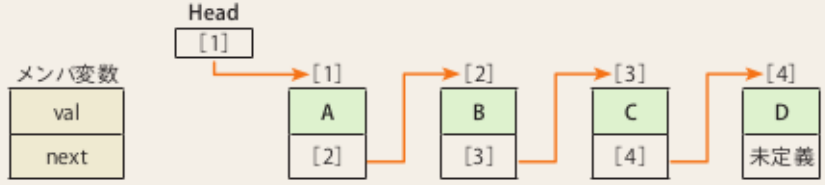
出典：2022年12月公開 基本情報技術者試験 科目Bサンプル問題 問10

問　単方向リストからの要素削除

単方向リストからの要素を削除する問題で、テーマとなるリストは、p.173の例題とほぼ同じです。念のため、仕様をまとめると次のようになります。

① クラスListElementは、単方向リストの要素を表す。
② ListElement型の変数はクラスListElementのインスタンスの参照を格納する。
③ 大域変数listHeadには、リストの先頭要素の参照があらかじめ格納されている。
④ メンバ変数nextは次の要素の参照が格納され、次の要素がないときの状態は未定義が格納されている。

手続delNodeについては、クラスのメンバ変数と使い方などもp.173の例題と同じで、異なるのはリスト要素を削除するという機能だけです。具体例として、下記のリストから要素3を削除することを想定してみましょう。

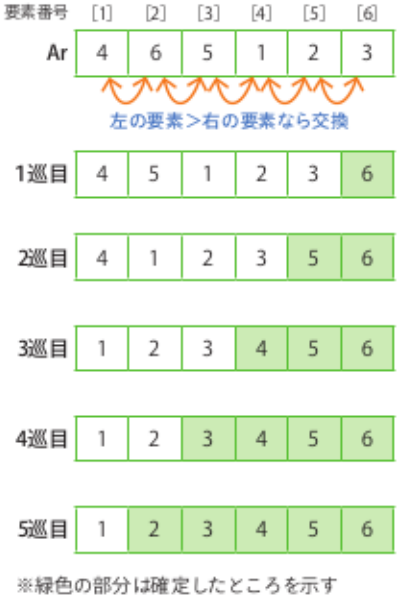


まず考えることは、「何を」、「どうやって削除するか」ということ。削除する要素の指定は、リストの先頭の位置を1とした整数値で受け渡されます。ただし、クラスListElementのメンバ変数には「要素番号」が含まれていないため、削除する要素がどれなのかわかりません。そこで、プログラムでは変数iを使って特定していきます。「どうやって削除」については、削除を行う要素から見て前側（先頭側）のポインタを変更します。プログラムでは、このメンバ変数の指定方法がカギになってきます。

バブルソートのアルゴリズム

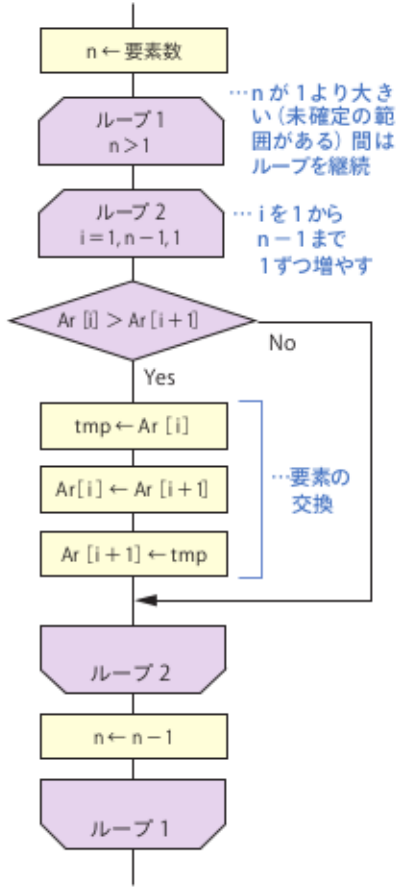
バブルソート（基本交換法）は、隣接する要素どうしの比較と入換えを繰り返し行うことで、すべての要素を整列する整列法です。配列Ar（Ar［ i ］: i ＝ 1, 2, 3, …, n）を昇順に整列するアルゴリズムを考えてみましょう。昇順の場合、1巡目のループ1の実行によって（i ＝ 1～（n－1）まで）、対象範囲内の最大値はAr［n］に格納されます。2巡目は比較・交換の範囲が一つ狭まり、Ar［n－1］に2番目に大きな値が格納されます。処理を繰り返すたびに、最上位の値が要素の末尾（または先頭）へ移動する様子を、浮き上がる泡に見立ててバブルソートと呼んでいます。

《バブルソートのトレース例（昇順）》



図表8-2-4　バブルソートのトレース

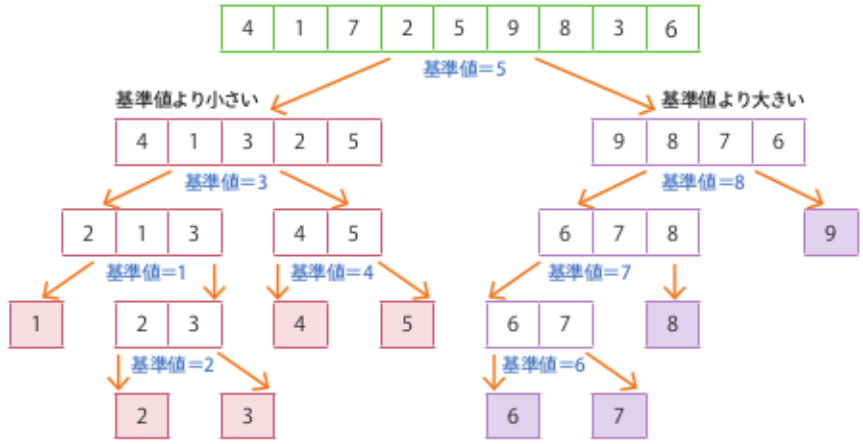
なお上のトレース例では、3巡目で整列は終了していますが、アルゴリズムは引き続き行われます。そこで、1度も交換が行われなかったことを示すフラグ（判定用の変数）をアルゴリズムに組み込むと、すでに整列済みに近い状態の配列なら、より短時間で整列できます。



図表8-2-5　バブルソートの流れ図

クイックソートのアルゴリズムに挑戦！

クイックソートは、基準値を決めて大きい値と小さい値に振り分け（基準値はどちらに入れても可）、さらにその中で基準値の大小に振り分けます。この振り分け動作を繰り返していき、最終的に要素が一つになれば整列が終了というアルゴリズムです。



図表8-2-6　クイックソートの動作（基準値は小さいほうへ含めている）

処理の概要をつかめたところで、クイックソートの問題に挑戦してみましょう。

例題

「クイックソートのアルゴリズム」

次の記述中の［　　］に入れる正しい答えを、解答群の中から選べ。ここで、配列の要素番号は1から始まる。

クイックソートは、指定された範囲のデータを、任意で選んだ基準値より小さい値のグループと基準値より大きい値のグループに分割し（基準値はどちらのグループに振り分けても構わない）、さらに、それぞれのグループの中で新たに基準値を選んで二つのグループに分割する処理を、グループの要素数が1になるまで繰り返すものである。

関数quickSortの引数で与えられた整数型配列arrayの内容が、{ 6, 2, 8, 5, 9, 3, 7 }、leftが1、rightが7のとき、プログラム中のα部分を2回目に実行したときの配列arrayの内容は{［　　］}になる。

解答群

	a	b	c
ア	arrayK[i]	arrayM[i]	allItems の要素数
イ	arrayK[i]	arrayM[i]	orders の要素数
ウ	arrayK[i]	arrayM[i]	otherItems の要素数
エ	arrayM[i]	arrayK[i]	allItems の要素数
オ	arrayM[i]	arrayK[i]	orders の要素数
カ	arrayM[i]	arrayK[i]	otherItems の要素数

出典：2024年7月公開　基本情報技術者試験科目B公開問題 問5

●強調フィルタリング（アイテムベース）の仕組み

強調フィルタリング方式は、レコメンドシステムを実現するための代表的なものです。レコメンド機能は、次の３段階で行っていきます。

①ショッピングサイトなどでの商品の購入履歴や閲覧履歴などの情報を収集する。

②収集した情報について、ルールを設けて特徴や共通点を抽出する。

③抽出結果を基に、特定の条件に適合した商品を提示する。

○処理の手順

問題文に提示されている注文データの例では、商品がA～Eの５つあるので、これを基に段階を踏んでクロス集計を行っていきます。

〔問題文〕購入された商品のリスト

注文番号	購入された商品のリスト
1	A, B, D
2	A, D
3	A
4	A, B, E
5	B
6	C, E

〔1〕注文番号ごとに購入商品を集計

注文番号	A	B	C	D	E
1	1	1		1	
2	1			1	
3	1				
4	1	1			1
5		1			
6			1		1

〔2〕商品軸ごとに同時購入商品を集計

商品軸＼同時に購入された商品	A	B	C	D	E
A		2		2	1
B	2			1	1
C					1
D	2	1			
E	1	1	1		

〔3〕計算式に基づいて関連度を集計

商品軸＼同時に購入された商品	A	B	C	D	E
A		1.0		1.5	0.75
B	1.0			1.0	1.0
C					3.0
D	1.5	1.0			
E	0.75	1.0	3.0		

〔4〕レコメンドする商品表

A	B	1.0
	C	
	D	1.5
	E	0.75
B	A	1.0
	C	
	D	1.0
	E	1.0
C	A	
	B	
	D	
	E	3.0
D	A	1.5
	B	1.0
	C	
	E	
E	A	0.75
	B	1.0
	C	3.0
	D	

収集した情報を基に、〔1〕～〔4〕のステップを踏むことで、レコメンドのベースとなる表が完成します。

○レコメンド処理の実際

表の数値はウエイトや確率など、レコメンド表示するルールに合わせて利用します。また問題文とは異なる計算式を使えば、値を調整できます。

例えば値の大きい順に提示するなら、商品Aを購入した人へ、商品D、商品Bを薦めることになります。ただし、左の表でもわかるとおり、情報を収集した際の購入数や同時に購入された数が少ないと値に偏りが出てしまいます(商品Cに関連する部分)。その場合は、別の仕組みを加えていく必要があるでしょう。

●プログラムを読み解いて空欄を考える

ここまで、テーマとなっているレコメンドシステムについて取り上げてきましたが、試験問題は短時間で解かなくてはならないので、仕組みを詳細に理解する必要はありません。空欄と解答群を見ると、関連度の計算式が、プログラムでどう実現されているかがポイントです。空欄cが計算式で、空欄aとbがその計算に使う値の集計方法であると見て取れます。計算式は提示されているので、空欄cから見ていきましょう。

○計算式で必要となる要素

問題文で提示された計算式とプログラムの次の部分を照らし合わせてみることで、どのような値が必要なのかがわかります。

$$L_{xy} = \frac{M_{xy} \times \text{全注文数}}{K_x \times K_y}$$

```
valueL ← (arrayM[i] × [ c ]) ÷ (itemCount × arrayK[i])
                                /* 実数として計算する */
```

- L_{xy} → valueL　…計算結果
- M_{xy} → arrayM[i]　…同一の注文で購入された注文数
- 全注文数　…空欄c