902063-01

「IT パスポート試験 シラバス」 Ver. 6.0 対応

擬似言語補足資料

（株）アイテック　ＩＴ人材教育研究部

目次

●この補足資料について

この資料は，2022 年 4 月から実施される IT パスポート試験で，プログラミング的思考力を問う擬似言語を用いた出題が行われる変更に対して，基本的な用語や擬似言語の記述方法の解説，サンプル問題の解説，演習問題を説明したものです。

プログラミング経験のない方は，用語の意味から確実に理解してください。

1　アルゴリズム

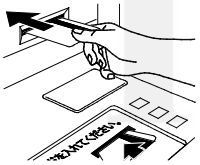
(1) アルゴリズムとは

コンピュータに，ある特定の目的を達成させるための処理手順のことをアルゴリズム（algorithm）といいます。少し細かくいうと，正しい結果を確実に得られたり，目的とする動作を確実に実行したりできる処理手順をアルゴリズムといい，正しい結果が得られるか分からないとか，目的とする動作が実行できるか分からないといった処理手順はアルゴリズムとはいいません。また，いつまで経っても処理が終わらない手順もアルゴリズムとはいいません。

アルゴリズムは正しい結果を確実に得るための処理手順ですが，「問題を解決するための手順」ともいえます。例えば，銀行の ATM でお金を引き出すときの行動を考えてみましょう。このときの問題は，「お金を受け取る」ことです。それを解決するためには，「キャッシュカードを入れる」，「暗証番号を押す」などの行動が必要です。

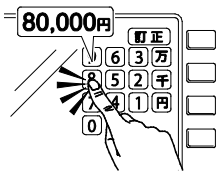
このときの行動を順番に表すと，次のようになります。

① ATM からお金を引き出すため，ATM メニューから「引き出し」を選択する。

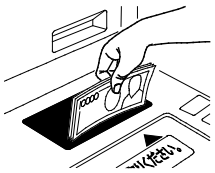


② キャッシュカードを入れる。

③ 暗証番号を押す。



④ 引き出す金額を入力する。



⑤ お金が出てくる（受け取る）。

この順序のとおりに行動すれば，お金を引き出すことができるので，この処理手順が銀行の ATM でお金を引き出すときのアルゴリズムといえます。

ここで，例えば，①と②の順序が入れ替わったら，キャッシュカードが挿入口から入らない機械もありますし，②と③が入れ替わったら，暗証番号を押しても ATM は反応しません。①，②，③，④，⑤と決められた順番に操作することで，はじめてお金を受け取ることができます。

このように，「しなければいけないこと（処理）を正しい順番で並べたもの」がアルゴリズムになります。コンピュータで計算などの処理を行う場合も同じで，どのようなデータを使って，どのような順番で計算をするかを正しい手順で指定したものがアルゴリズムとなります。

(2) アルゴリズムの例

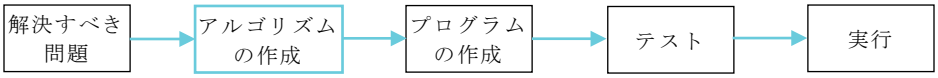
ATM でお金を引き出すアルゴリズムを見ましたが，他にもいろいろな例があります。例えば，朝起きてから仕事に行ったり学校に行ったりするまでの行動や，自動販売機でコーヒーを買うまでの手順などが挙げられます。

このように日常でしている行動のアルゴリズムもありますが，コンピュータに行わせる処理を考えると，一番得意なことは計算といえます。多くのデータを集計して最新の結果を求めたり，過去の結果を基に天気の変わり方を予測したりする処理では，結果だけを見ても分かりませんが，コンピュータ内部では決められたアルゴリズムに従って，膨大な計算処理が超高速で実行されているのです。

みなさんが受験する IT パスポート試験でも，これまでに少しですがアルゴリズムの問題が出題されていました。値の合計を求めたり，データを比較して大小を判断したりというごく簡単な問題がほとんどでした。

(3) アルゴリズムからプログラムへ

アルゴリズムはコンピュータに対する指示書になるので，アルゴリズムができた後はこの指示書に基づいてプログラムの作成を行います。



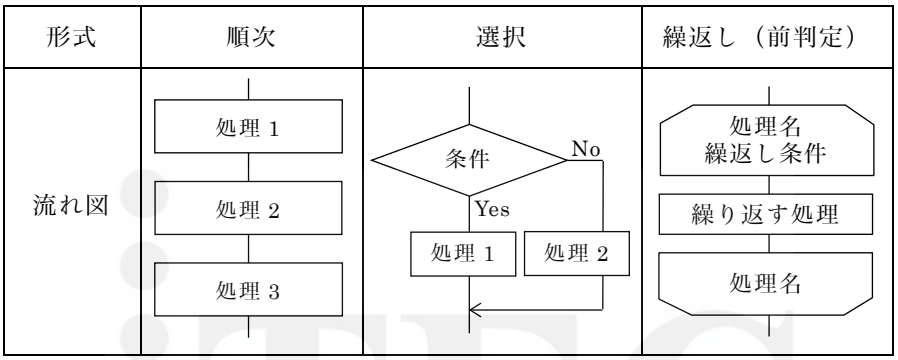
この補足資料で説明するのは，この中の「アルゴリズムの作成」の部分で，基本的な用語の意味と擬似言語の表記方法を理解した後，IPA から発表されたサンプル問題の解説で擬似言語によるプログラムの読み方を把握し，練習問題で実力をつけます。

学習に当たって大切なことは，「なるほど！」と思えるまで自分の頭で考えることです。めんどうに思えるときもあるかもしれませんが，この気持ちを忘れずに一歩一歩学習していきましょう。

(4) アルゴリズムの表現方法

アルゴリズムは，これまでの IT パスポート試験では流れ図で表現されることが多かったのですが，今回の出題内容の見直しによって，擬似言語を使った方法でアルゴリズムの問題が出題されることになりました。流れ図は記号で処理内容を表現しますが，擬似言語では選択の `if` や繰返しの `for`，`while`，`do` といった単語（プログラムの命令に相当します）で処理内容を表現します。

アルゴリズムの表現方法について，次の三つの基本処理（順次，選択，繰返し）を組み合わせることによって，全てのアルゴリズムが表現できることが知られています（ダイクストラ，構造化定理）。



アルゴリズムを構成する三つの基本処理

この後で擬似言語の書き方を詳しく見ていきますが，「順次」は一連の処理のつながりを表しています。「選択」は複数の条件を指定できるようになっています。また，「繰返し（前判定）」では，始めに繰返し条件の判定を行い，条件を満たしていれば処理を実行する繰返し処理を記述できるようになっています。

擬似言語自体はプログラミング言語の C と似た表現方法になっていて，擬似言語でアルゴリズムを考えたら，C のプログラムに置き換えやすい仕様になっているといえます。なお，IT パスポート試験では具体的なプログラミング言語による問題は出題範囲に入っていないので，詳しい説明は省略します。

(5) アルゴリズムと擬似言語に関連する基本的な用語

① 手続，関数

ある目的を実現するための一連の処理を「手続」や「関数」といいます。このうちの「関数」は個別の処理を行う機能をもち，処理に必要なデータ（引数という）を指定して関数を実行すると，処理結果（戻り値）を返す（return）という特徴があります。

「手続」も個別の処理を行いますが，処理結果を出力する場合・出力しない場合も含

めて，指定した一連の処理を順番に実行していくと考えると，関数との違いが分かりやすいでしょう。

擬似言語の試験問題で手続や関数が出題されるときは，処理するデータや処理の内容について問題文で説明されるので，内容をしっかり理解するようにしましょう。

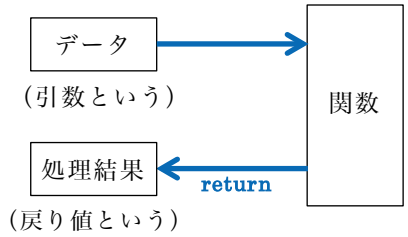


図　関数のイメージ

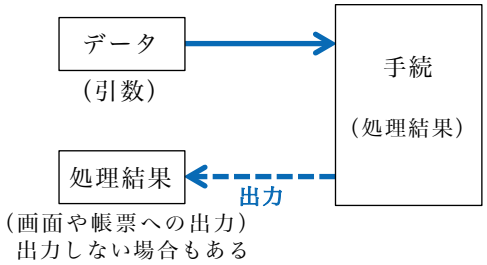


図　手続のイメージ

②　プログラム

情報処理の用語としては，手続や関数の中で規則に従って指定された処理内容を示す用語として，「プログラム」を使うことが多いといえます。

③　データ

コンピュータを使った処理では，様々なデータを使います。例えば，計算するためのデータ，計算途中の結果を格納しておくデータ，最終結果を格納するデータ，など様々です。これらは目的ごとに別々のデータとして名前を付けてプログラムの中で示します。

④　定数

データの中で，処理を通じて値が変わらない（変えない）データのことを「定数」といいます。円周率は定数の代表的なものですが，個々の処理に必要な定数はプログラムの中で複数指定することができます。

⑤　変数

データを処理する過程で値が変わるデータを「変数」といいます。途中結果を格納するデータや，同じ処理を繰り返すときの回数を示すデータなどが変数に当たります。処理に必要な変数はプログラムの中で幾つでも指定することができ，それぞれ別の名前を付けて宣言します。

⑥　引数

関数に処理させるデータを「引数（ひきすう）」といいます。処理した結果を格納するデータを引数に含める場合もあります。関数を呼び出す側との間でやり取りするデータといえます。

⑦　型

データの種類（属性）を表す用語です。「データ型」というように続けて表現することもあります。データの型によって，コンピュータのメモリ上に確保されるデータの長さや，計算の際に演算装置内で動作する演算回路が異なるため，プログラムの中で使うデータは，名前の他にこの型も指定する必要があります。

具体的なデータの型には，数値データの整数型や実数型，文字データの文字型，条件の指定で使える論理型などがあり，プログラムで使うデータ（定数や変数）について，それぞれの型を示します（データの型を宣言するといいます)。

なお，論理型は，真（true）と偽（false）のどちらかの意味を示すデータのことです。true と false はこの綴りの単語自体が擬似言語の中で特別な定数を表しているので，変数の名前としては使えません。

⑧　配列

変数や定数を使う宣言をすると，データを格納する領域がメモリ中に確保されます。例えていうと，データを入れるための“箱”が用意されるといえます。この同じ箱を連続して複数つなげたものが配列です（メモリ中に連続した領域が確保されます)。

配列は同じ型のデータが連続しているので，どのデータかを示すために，先頭から何番目かを表す「要素番号」を指定します。

参考までに配列の英語は array で，後で掲載するサンプルプログラムの配列名で使われています。

⑨　代入

数値や文字を変数や定数に格納することを「**代入**」といいます。値が格納されている変数や定数を，別の変数や定数に代入することもできます。

⑩　条件

アルゴリズムの中で指定する処理は，必ず実行するもの以外に，特定の条件を満たすときだけ実行するものも指定できます。条件の書き方は，「T が 100 以上のとき」のように具体的な言葉で示したり，「T ≧ 100」のように，演算子を使って表したりすることができます。また，複数の条件を and（かつ）や or（または）で結び付けて，複雑な条件を指定することもできます。

⑪　演算子

計算式や条件を指定するときに使える演算内容や記号を「**演算子**」といいます。加減乗除の演算内容を指定する“＋－×÷”や，余りを求める剰余算の“mod”，式のまとまりを示すために使う“()”，大小関係を表す“≠≦≧＜＝＞”，複数の条件を指定すると

きに使う論理演算の and，or などがあります。
・単項演算子……正負の符号や否定など，一つの項に付ける演算子
・二項演算子……二つの項に対する加減乗除の演算や大小関係の等号・不等号など

⑫　式

プログラムの中で使用する定数などの値や変数，演算子などを組み合わせた記述を「**式**」といいます。複数の組合せではなく，一つの値や変数だけでも式に含まれます。条件などを指定するための式を特に**条件式**といいます。

⑬　文

プログラムの中で実行する一つの処理の記述を「**文**」といいます。式は文を構成する一部の要素です。IT パスポート試験用の擬似言語には，代入文や手続・関数を呼び出す文，`if` 文，`while` 文，`do` 文，`for` 文があります。

（関数名，手続名，変数名の付け方に関する補足）

関数名や手続名は処理内容の分かる名前が付けられます。また変数はデータの用途が分かるような名前が付けられますが，単純な繰返し回数を示す変数の場合は，`i` や `j` などの 1 文字の変数名が付けられることが多いです。

なお参考までに，情報処理技術者試験の問題では英語で意味の分かる名前が付けられることが多いです。例えば，合計を求める変数では gokei のような日本語読みの名前ではなく，sum のように英語で意味の分かる名前が使われます。日本語を知らない人への配慮がされていると考えられます。

2　擬似言語

アルゴリズムを表現する代表的な方法が流れ図ですが，実際のプログラミング言語に近い書き方で表現できるのが擬似言語です。IT パスポート試験で使われる擬似言語は決まりも少ないので，基本事項を理解しておきましょう。

なお，これまでの参考資料の使い方から想定して，試験中でも必要に応じて見られる資料になると考えられますが，問題を解くことに試験中はなるべく専念した方がいいので，事前に擬似言語の内容をしっかり理解しておくことが大切です。

IPA から発表された IT パスポート試験用の擬似言語全体は「5.（付録）擬似言語の記述形式（IT パスポート試験用）」で掲載しますが，ここでは記載されている個々の内容について，〔擬似言語の記述形式〕，〔演算子と優先順位〕，〔論理型の定数〕，〔配列〕の順に資料に記述されている説明とその補足をしていきます。

（1）擬似言語の記述形式

① ***○手続名又は関数名***

（説明）手続又は関数を宣言する。

（補足）プログラムの最初に手続又は関数の名前を示します。処理するデータである引数は名前の後ろに（ ）の中で指定します。関数の場合は，処理結果のデータである戻り値の型を最初に指定します。

（例 1）**○実数型：** `calcMean(`**実数型の配列：** `dataArray)`

関数名が `calcMean` で，実数型の配列 `dataArray` を引数として受け取り，処理を行って，実数型の結果を返すことを示します。

（例 2）**○整数型：** `calc_factorial(`**整数型：** `num)`

関数名が `calc_factorial` で，整数型の数値 `num` を引数として受け取り，階乗（英語で factorial）の計算を行って，整数型の結果を返すことを示します。

（例 3）**○**`printStars(`**整数型：** `num)`

手続名が `printStars` で，整数型の数値 `num` を引数として受け取り，処理を行うことを示します。○の後ろに型がないので，関数ではなく手続になります。

② ***型名: 変数名***

（説明）変数を宣言する。

（補足）手続や関数の宣言に続いて，処理で使う変数の型と名前を示します。

（例 1）**実数型：** `sum, mean`

実数型の変数として `sum` と `mean` を使うことを宣言

（例 2）**整数型：** cnt ← 0

整数型の変数として cnt を使う宣言と，初期値 0 の設定を併せて行う例

③ ***/* 注釈 */***
// 注釈

（説明）変数を宣言する。

（補足）プログラムには，後で見たときに処理内容やアルゴリズム，注意事項を分かりやすく示すために注釈を入れられます。試験用の擬似言語では，「/* と */ で囲まれた記述」や，「// の後ろの記述」は注釈として処理の実行に影響しません。なお，試験問題ではヒントになり過ぎる注釈は省略される場合があります。

（例）**/* 実数として計算する */**
// 実数として計算する

④ ***変数名 ← 式***

（説明）変数に***式***の値を代入する。

（補足）←の方向に従って，代入先を左側に記述し，代入する数値や変数，計算式などを示します。

（代入先）←（代入する値や変数などの式）

（例 1）sum ← 0

変数 sum に 0 を代入する

（例 2）cnt ← cnt ＋ 1

現在の cnt の値に 1 を足した値を，改めて cnt に代入する。

現在の cnt の値が 5 なら，1 を足した 6 が cnt に代入される。

⑤ ***手続名又は関数名(引数，…)***

（説明）手続又は関数を呼び出し，***引数***を受け渡す。

（補足）プログラムの中で手続名や関数名を指定すると，それらの手続や関数が実行されます。受け渡す引数は手続名や関数名の後ろに（）の中で指定します。複数の引数がある場合は，“,”で区切って指定し，引数は呼び出す側で指定した引数の順に，各手続や関数で指定した引数に対応します。

（例）（関数の宣言）　**○整数型：** calc(整数**型：** num1，整数**型：** num2)

ある手続の中でこの関数 calc を呼び出して実行する。

（呼出し時）　calc(w_a，w_b)

呼び出した時点の w_a，w_b の値を引数として，この順に関数 calc の引数 num1，num2 の値となって関数が実行される。

⑥

```
if (条件式1)
  処理1
elseif (条件式2)
  処理2
elseif (条件式n)
  処理n
else
  処理n＋1
 endif
```

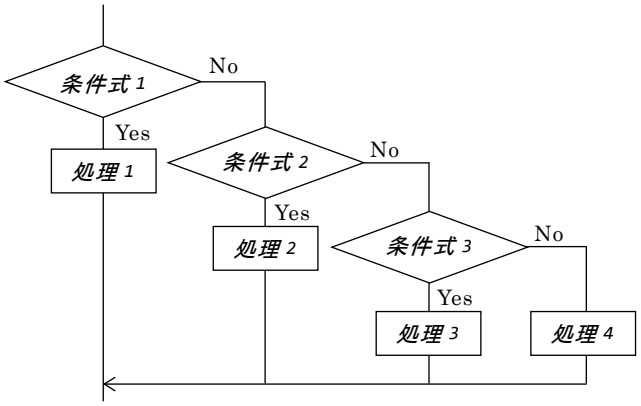


図　条件式が三つある場合の流れ図

（説明）選択処理を示す。

条件式を上から評価し，最初に真になった***条件式***に対応する***処理***を実行する。以降の***条件式***は評価せず，対応する***処理***も実行しない。どの***条件式***も真にならないときは，***処理***$_{n+1}$ を実行する。

各***処理***は，0 以上の文の集まりである。

elseif と***処理***の組みは，複数記述することがあり，省略することもある。

else と***処理***$_{n+1}$ の組みは一つだけ記述し，省略することもある。

（補足）条件式は幾つでも指定できますが，条件が一つの場合は次のように 2 分岐の if 文になります。***処理***$_1$ か***処理***$_2$ のどちらかの文が実行されます。

```
if (条件式1)
  処理1
else
  処理2
endif
```

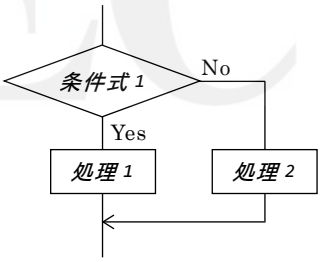


図　条件式が一つの場合の選択処理の流れ図

この if 文で else と***処理***$_2$ を省略して，***条件式***$_1$ が真のときに実行する処理だけを記述することもできます。

```
if (条件式1)
  処理1
endif
```

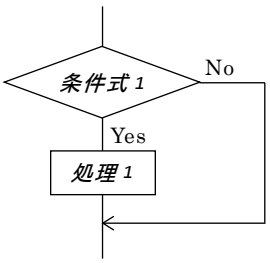


図　else と*処理*$_2$ を省略した場合の流れ図

（プログラム例 1）

・処理　整数型の引数 A, B を受け取り大小比較を行って，メッセージを出力する。
　A が B 以上のとき，「データ A の値はデータ B 以上です」を出力する。
　A が B より小さいとき，「データ A の値はデータ B より小さいです」を出力する。

・プログラム（手続の例）

```
○data_judge(整数型: A, 整数型: B)
                        /* 手続 data_judge の宣言（引数は整数型の A と B） */
  if (A ≧ B)  <――― 条件式
    "データ A の値はデータ B 以上です"を出力する  <――― 処理 1
  else
    "データ A の値はデータ B より小さいです  <――― 処理 2
  endif
```

（プログラム例 2）

・処理　整数型の引数 points を受け取り，個別のメッセージを出力する。
　points が 80 以上のとき，「A ランクです」を出力する。
　points が 60 以上 80 未満のとき，「B ランクです」を出力する。
　points が 40 以上 60 未満のとき，「C ランクです」を出力する。
　points が 40 未満のとき，「D ランクです」を出力する。

・プログラム（関数の例）

```
○整数型: judgePoints(整数型: points)    /* 関数 judgePoints の宣言 */
  if (points ≧ 80)  <――― 条件式
    "A ランクです"を出力する  <――― 処理 1
  elseif (points ≧ 60)
    "B ランクです"を出力する  <――― 処理 2
  elseif (points ≧ 40)
    "C ランクです"を出力する  <――― 処理 3
  else
    "D ランクです"を出力する  <――― 処理 4
  endif
```

⑦

```
while (条件式)
      処理
endwhile
```

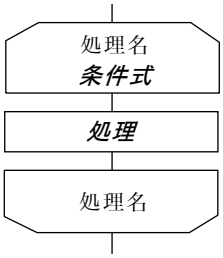


図　前判定繰返し処理の流れ図（while）

（説明）前判定繰返し処理を示す。

条件式が真の間，***処理***を繰返し実行する。

処理は，0 以上の文の集まりである。

（補足）繰返し処理を実行する前に，繰り返す条件式の判定を行い，真の間，処理を繰返し実行します。条件式の判定で偽になったら，繰返し処理を抜けて次の文を実行します。最初から条件式が偽の場合は，一度も処理を実行せずに繰返し処理を抜けて，endwhile の次の文を実行します。

（プログラム例）

・処理　整数の 1 から 100 までの合計を求めて出力する。

・アルゴリズム　整数の i と sum を変数として使い，i の初期値を 1 として，合計を sum に求める。

・プログラム（手続の例）

```
○calcSum()                  /* 手続 calcSum の宣言（引数なし） */
  整数型: i ← 1             /* 加算する変数 i の宣言と初期化 */
  整数型: sum ← 0           /* 合計を求める変数 sum の宣言と初期化 */
  while (i ≦ 100)           /* i の値が 100 以下なら処理を実行 */
    sum ← sum + i           /* i の値を sum に加算 */
    i ← i + 1               /* i の値に 1 を加算 */
  endwhile
  sum の値を出力する
```

処理

（プログラムの字下げについて）

if 文，for 文，while 文，do 文の中で指定する処理は，記述する先頭位置を右にずらして読みやすくします。これを字下げといいます。字下げする文字数は擬似言語では決められていませんが，サンプルプログラムでは，2 文字右にずらして記述しています。自分で擬似言語のプログラムを書くときは，この 2 文字の字下げを行うようにしましょう。

⑧

```
do
  処理
while (条件式)
```

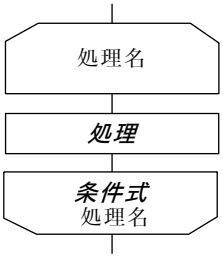


図　後判定繰返し処理の流れ図(do)

(説明) 後判定繰返し処理を示す。

処理を実行し，***条件式が***真の間，***処理***を繰返し実行する。

処理は，0 以上の文の集まりである。

(補足) 繰返し処理を 1 回実行した後に，繰り返す条件式の判定を行い，真の間，処理を繰返し実行します。条件式の判定で偽になったら，繰返し処理を抜けて，while (***条件式***)の次の文を実行します。最低 1 回は処理を実行させたい場合に，この do 文を使います。

(プログラム例)

・処理　整数型の引数 num を受け取り，整数の 1 から num の値までの合計を sum に求めて戻り値として返す。

・アルゴリズム　整数の i と sum を変数として使い，i の初期値を 1 として，num まで加算しながら，合計を sum に求める。

・プログラム (関数の例)

```
○整数型: calcSum(整数型: num)    /* 関数 calcSum の宣言 */
  整数型: i, sum    /* 変数 i と sum の宣言 */
  i ← 1          /* i を 1 で初期化する */
  sum ← 0        /* sum を 0 で初期化する */
  do
    sum ← sum + i    /* i の値を sum に加算 */
    i ← i + 1        /* i の値に 1 を加算 */
  while (i ≦ num)   /* i の値が num 以下なら処理を実行 */
  return sum      /* sum の値を関数の呼出し元に戻り値として返す */
```

処理

・(注意) 誤って，i を引数 num の値よりも大きな値で初期設定した場合でも，i の値を sum に加算する処理が 1 回実行されてしまいます。

⑨

```
for (制御記述)
  処理
endfor
```

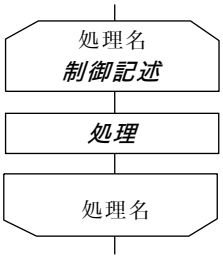


図　for 文による繰返し処理の流れ図

（説明）繰返し処理を示す。

制御記述の内容に基づいて，***処理***を繰返し実行する。

処理は，0 以上の文の集まりである。

（補足）考え方は while 文の前判定繰返し処理と同じですが，処理を一定回数だけ単純に繰り返す場合などに，for 文がよく使われます。なお，制御記述の方法については，「擬似言語の記述形式」の中で説明がないので，IT パスポート試験ではサンプル問題として公表されたプログラムにあるような「～を～～まで増やす」というような特定の変数を増減させる制御記述になると思われます。

（プログラム例）

・処理　整数型の引数 num を受け取り，整数の 1 から num の値までの合計を sum に求めて戻り値として返す。

・アルゴリズム　整数の i と sum を変数として使い，i の初期値を 1 として，num まで加算しながら，合計を sum に求める。

・プログラム（関数の例）

```
○整数型: calcSum(整数型: num)    // 関数 calcSum の宣言
  整数型: i, sum    // 変数 i と sum の宣言
  sum ← 0      // sum を初期化
  for (i を 1 から num まで 1 ずつ増やす)  // i が num を超えたら endfor の次へ
    sum ← sum + i    // i の値を sum に加算      処理
  endfor
  return sum      // sum の値を関数の呼出し元に戻り値として返す
```

（注意）for 文の制御記述の中に「i の値に 1 を加算する処理」を含んでいるので，「i の値を sum に加算する処理」だけ繰り返す処理として記述すればよく，プログラムを簡素化できます。

(2) 演算子と優先順位

擬似言語で指定できる演算子を優先順位の高い順に説明します。優先順位は複数の演算子を式の中で記述したとき，どの演算子を先に実行するかという順序を決めたものです。実際のプログラム言語で使われる演算子の種類・優先順位とほとんど同じと考えてください。

① **式　()**

（補足）条件式の記述や，長い式の部分的なまとまりを示すのに使われる括弧です。()の中にさらに()を使って式を示すことができますが，複数指定されている場合は，どこの“(”と“)”が対応しているかをしっかり把握する必要があります。

（例 1）条件式　(A が 10 より大きい)

（例 2）式　(A ＋ B)×(C － D)

② **単項演算子　not ＋ －**

（補足）一つの数値や変数に付ける演算子で，not は論理型の変数を否定します。
＋と－は，数値や変数の正負を表す符号として使います。

（例 1）論理型変数：P に対して，not P
（P は真偽を表す true か false のどちらかで,その否定を not P と記述します）

（例 2）変数や数値の符号　＋B　－2　など。括弧を付けて(＋B)　(－2)として単項演算子であることをはっきり示す場合もあります。

③ **二項演算子　乗除　mod × ÷**

（補足）二つの数値や変数などの乗除算で使う演算子です。除算は／ではなく，÷を使います。mod は余りを求める剰余残の演算子で，例として，20 mod 6 は 2 となります（20 を 6 で割ると，商が 3 で，余りが 2）。

（例 1）代入文で，C ← A × B　　D ← (A ＋ B)÷2

（例 2）条件式で，while (B ≦ 100)　（変数 B が 100 以下なら処理を繰り返します）

（例 3）剰余残　C ← A mod B　（A を B で割った余りを C に代入します）

④ **二項演算子　加減　＋　－**

（補足）二つの数値や変数などの加減算で使う演算子です。単項演算子と混在する場合は()を付けて混乱しないように指定します。

（例 1）代入文で，C ← A ＋ B

（例 2）代入文で，C ← A ＋ (－B)

⑤ **二項演算子　関係　≠ ≦ ≧ ＜ ＝ ＞**

（補足）二つの数値や変数などの大小や，等しい・等しくないといった関係を示すときに使う演算子です。≦ は「＜ または ＝」，≧は「＞ または ＝」の意味になり，＝ の場合も含んでいます。

（例 1）if 文の条件式で，if (C ≠ A)　　（C と A が等しくない）

（例 2）while 文の条件式で，while (P ＜ Q)　　（P が Q より小さい）

⑥ **二項演算子　論理積　and**

（補足）条件式の指定で，「二つの条件の両方が真のときに，条件全体を真とする」ような指定をするときに使う論理演算の演算子です。

（例）if (A ≦ 10) and (A ＝ K)　　（A が 10 以下で，かつ，A と K が等しい）とき

真となる場合……A＝7 で K＝7 のとき，A＝10 で K＝10 のとき

A＝3 で K＝3 のとき，など

偽となる場合……A＝11 で K＝11 のとき，A＝3 で K＝2 のとき

⑦ **二項演算子　論理和　or**

（補足）条件式の指定で，「二つの条件の少なくともどちらか一方が真のときに，条件全体を真とする」ような指定をするときに使う論理演算の演算子です。このとき，両方の条件が真のときにも条件全体は真となります。

（例）if (A ≦ 10) or (A ＝ K)　　（A が 10 以下，または，A と K が等しい）とき

真となる場合……A＝7 で K＝10 のとき，　A＝12 で K＝12 のとき　など

偽となる場合……A＝11 で K＝12 のとき

（補足）　演算子が混ざっている例

演算子が混ざっている場合は優先順位の高いところから演算されますが，混乱を避けるために( )を付けることが多いです。

（例 1）代入文　ANS ← A × B ＋ C

（乗算の×の方が，加算の＋よりも優先順位が高いので，A と B を乗算した結果に C を加算します）

※プログラムを読む人が誤って解釈しないように，ANS ← (A × B) ＋ C　と記述することがあります。

（例 2）条件　if (A ≦ B or C ＞ 100 and D ＝ 1)

※　関係を示す≦，＞，＝が，論理演算の and や or より優先順位が高いので，まず，「A ≦ B」，「C ＞ 100」，「D ＝ 1」と結び付きます。

次に，論理演算の論理積 and の方が，論理和 or よりも優先順位が高いので，条件式 「C ＞ 100 and D ＝ 1」の真偽を調べた後で，条件 「A ≦ B」の真偽との or の結果を調べます。

このように長い条件式の記述をすると，プログラムを作る人も読む人も誤って解釈しやすいので，if (A ≦ B) or (C ＞ 100 and D ＝ 1) のように() を付けて記述することが多いです。

(3) 論理型の定数（true，false）

論理型の変数は真と偽の二つの状態をもつ変数です。真の状態にするには論理型の定数 true を代入し，偽にするには論理型の定数 false を代入します。これらの論理型定数 true と false は，あらかじめ決められた定数として特別扱いされ（予約語といいます），この綴りと同じ変数名，手続名，関数名は指定できません。

次のように論理型の変数を定義して，簡潔に条件を書くことができますが，使い方が少し難しいので，IT パスポート試験で出題されるかどうかは分かりません。

（使い方の例）

実数型のデータ samp の値が 1.25 以上のとき論理型変数 status を真にして，samp の値が 1.25 未満のとき status を偽にする。その後，status が真なら「Good！」を，偽なら「Bad！」を出力する。

```
実数型: samp
論理型: status
if (samp ≧ 1.25)
  status ← true        // samp ≧ 1.25 のとき，status を真にする
else
  status ← false       // samp ＜ 1.25 のとき，status を偽にする
endif
  :   （途中略）
if (status)
  "Good！"を出力する     // status が真（samp ≧ 1.25）のとき
else
  "Bad！"を出力する      // status が偽（samp ＜ 1.25）のとき
endif
```

※「samp ≧ 1.25」という条件式を，一つの論理型変数 status で簡潔に表すことができ，長い条件式が何度も出てくる場合に，同じ条件を繰返し書かずに済む利点があります。

(4) 配列

この資料の P.5「アルゴリズムと擬似言語に関連する基本的な用語」で配列の概要について説明しましたが，同じ型が繰返し連続して格納されるデータのことです。個々のデータは，配列名に要素番号を付けて指定します。

IPA の資料では，次のように説明されています。

〔配列〕

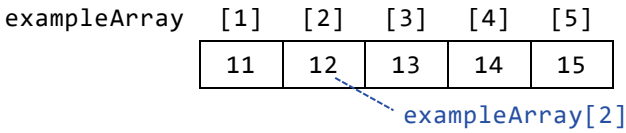
一次元配列において“{”は配列の内容の始まりを，“}”は配列の内容の終わりを表し，配列の要素は，“[”と“]”の間にアクセス対象要素の要素番号を指定することでアクセスする。

（例）要素番号が 1 から始まる配列 exampleArray の要素が{11, 12, 13, 14, 15}のとき，要素番号 4 の要素の値（14）は exampleArray[4]でアクセスできる。

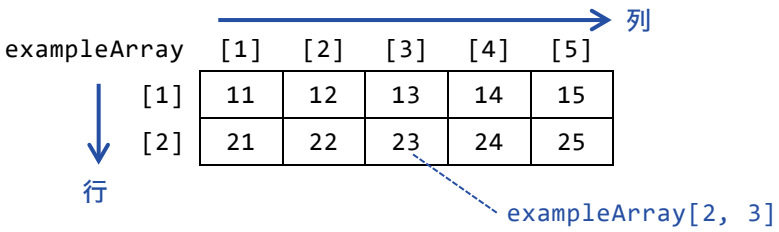
二次元配列において，内側の“{”と“}”に囲まれた部分は，1 行分の内容を表し，要素番号は，行番号，列番号の順に“,”で区切って指定する。

（例）要素番号が 1 から始まる二次元配列 exampleArray の要素が
{{11, 12, 13, 14, 15}, {21, 22, 23, 24, 25}}のとき，2 行目 5 列目の要素の値（25）は，exampleArray[2, 5]でアクセスできる。

この説明に出てくる一次元配列 exampleArray にデータが格納されているイメージを図で表すと，次のようになります。要素番号はプログラム言語によっては 0 から始まるものもありますが，試験用の擬似言語では 1 から始まる仕様になっています。



次に出てくる二次元配列 exampleArray に縦横に要素を格納する表をイメージすると分かりやすいです。データが格納されているイメージを図で表すと，次のようになります。図の下方向の要素番号が行に，右方向の要素番号が列に相当し，要素を指定するには，配列名[行番号，列番号] と指定します。



3　サンプル問題の解説

IT パスポート試験の擬似言語サンプル問題を解説します。問題を解くに当たっては，まず自分自身でプログラムの 1 行ごとの宣言や処理内容を考え，自分用の注釈を加えて処理内容を理解してください。そして，自分で解答を出したら解説などを読むようにしましょう。

(1) サンプル問題　問 1（関数）

問 1　関数 calcMean は，要素数が 1 以上の配列 dataArray を引数として受け取り，要素の値の平均を戻り値として返す。プログラム中の a，b に入れる字句の適切な組合せはどれか。ここで，配列の要素番号は 1 から始まる。

〔プログラム〕

```
○実数型: calcMean(実数型の配列: dataArray) /* 関数の宣言 */
  実数型: sum, mean
  整数型: i
  sum ← 0
  for (i を 1 から dataArray の要素数まで 1 ずつ増やす)
    sum ← [ a ]
  endfor
  mean ← sum ÷ [ b ]  /* 実数として計算する */
  return mean
```

	a	b
ア	sum ＋ dataArray[i]	dataArray の要素数
イ	sum ＋ dataArray[i]	(dataArray の要素数 ＋ 1)
ウ	sum × dataArray[i]	dataArray の要素数
エ	sum × dataArray[i]	(dataArray の要素数 ＋ 1)

（問 1 の解説）

プログラムを見て，まず引数の型と名称を確認すると，実数型の配列 dataArray で，関数 calcMean の処理が，配列要素の平均を計算して戻り値を実数型の mean として返す処理であることが分かります。次に要素数が 5 の dataArray の例として，次のような図をメモすると処理が分かりやすくなります。

dataArray	[1]	[2]	[3]	[4]	[5]
	11.2	10.5	13.1	15.6	12.8

ここで，平均を求めるには要素の合計を求めて，要素数で割ればよいので，プログラムのどこでその処理を行っているかを考えると，合計を求める変数が実数型の sum で配列要素の先頭（要素番号 1 の dataArray[1]）から最後の要素（dataArray[要素数]）まで加算しているのが，空欄 a の処理と分かります。

for 文の制御記述を見ると「i を 1 から dataArray の要素数まで 1 ずつ増やす」となっていることから，要素番号を整数型の i としていることが分かります。したがって，空欄 a には sum ＋ dataArray[i] が入ることが分かります。

次に for 文の実行を終えると，sum に合計が求められているので，この sum を配列 dataArray 要素数で割って平均を求める処理が空欄 b と分かります。

以上から，空欄 a を sum ＋ dataArray[i] ，空欄 b を配列 dataArray の要素数としている（ア）が正解になります。

なお，先ほど例で示した配列 dataArray の要素{11.2, 10.5, 13.1, 15.6, 12.8}でこのプログラムを実行すると，(11.2＋10.5＋13.1＋15.6＋12.8}÷5＝63.2÷5＝12.64 となり，mean の値 12.64 が戻り値として返されます。

正解　ア

（参考）IPA 資料でのこの問題の出題主旨

問番号	出題主旨
問 1	与えられたデータの平均を求める処理を題材として，配列を理解する能力，及び平均を求めるための計算を行うアルゴリズムをプログラム（関数）で表現する能力を問う。

(2) サンプル問題　問 2（手続）

問２　手続 printStars は，“☆”と“★”を交互に，引数 num で指定された数だけ出力する。プログラム中の a，b に入れる字句の適切な組合せはどれか。ここで，引数 num の値が 0 以下のときは，何も出力しない。

〔プログラム〕

```
○printStars(整数型: num)       /* 手続の宣言 */
  整数型: cnt ← 0               /* 出力した数を初期化する */
  文字列型: starColor ← "SC1"  /* 最初は“☆”を出力させる */
  [ a ]
    if (starColor が "SC1" と等しい)
      "☆"を出力する
      starColor ← "SC2"
    else
      "★"を出力する
      starColor ← "SC1"
    endif
    cnt ← cnt + 1
  [ b ]
```

	a	b
ア	do	while (cnt が num 以下)
イ	do	while (cnt が num より小さい)
ウ	while (cnt が num 以下)	endwhile
エ	while (cnt が num より小さい)	endwhile

（問 2 の解説）

手続 printStars の引数は整数型の num で，num の数だけ☆と★を交互に出力します。次にプログラムの注釈を参考にすると，出力した数を整数型の変数 cnt に格納していること,最初は星を出力していないので 0 で cnt を初期化していることが分かります。そして，☆と★のどちらを出力するかを示すデータ（"SC1"か"SC2"）を格納する変数が文字列型の starColor になっているという具合に，処理の概要を把握します。

次に空欄に入る字句の選択肢を見ると，do，while (条件式)，endwhile の組合せにな

っているので，前判定か後判定のどちらの繰返し処理を使うかを考えることになります。プログラムの 3 行目を見ると，「最初は“☆”を出力させる」という注釈があり，starColor に”SC1”を代入しています。このため，空欄 a に do 文が入るとすると，引数 num の値に関係なく次の処理 if 文が実行され，条件式(starColor が"SC1"と等しい)が真なので☆が出力されます。しかし，「引数 num の値が 0 以下のときは，何も出力しない」と問題にあるので正しい処理結果でないことになります。このことから空欄 a には while (条件式)，空欄 b には endwhile が入る（ウ）か（エ）が正解候補になります。

ここで，if 文で繰り返す処理の内容を見ておくと，starColor が"SC1"と等しいときは“☆”を出力し，そうでないとき（starColor が"SC2"のとき）は“★”を出力する処理になっていることが分かります。また，☆と★を交互に出力するために，“☆”を出力したら starColor に"SC2"を代入し，“★”を出力したら starColor に"SC1"を代入して，次の出力の準備をしています。

プログラムの処理を続けて見ていくと，星を出力する if 文の終わりを示す endif の次に，「cnt ← cnt ＋ 1」があります。cnt の初期値は 0 で，1 文字目の☆を出力したら cnt の値が 1 になり，次に 2 文字目の★を出力したら cnt の値が 2 になり，3 文字目の☆を出力したら cnt の値が 3 になる，……というように，星を出力した後で，出力文字数の cnt を更新していることに注意します。

この考え方から，例えば num＝3 でこのプログラムを実行したとき，3 文字目の☆を出力した後に cnt の値が 3 になるので，cnt の値が num と等しくなったら繰返し処理を終了し，cnt の値が num より小さいときは処理を繰り返すことになり，処理を繰り返す条件式は，(cnt が num より小さい)の（エ）が正解になります。

まとめとして，引数 num の値で出力される内容を示すと，次のようになります。

num	出力結果
0	なし
1	☆
2	☆★
3	☆★☆
4	☆★☆★
5	☆★☆★☆
:	:

（注）ここでは，文字を出力すると次の出力位置が右にずれると仮定しています。

正解　エ

（参考）IPA 資料でのこの問題の出題主旨

問番号	出題主旨
問 2	問題文に示された仕様を理解し，出力結果をイメージできる能力，及び それを満たす繰返し処理，選択処理から成るアルゴリズムをプログラム（手続）で表現する能力を問う。

4　練習問題

サンプル問題を参考に作成した練習問題を 2 問掲載します。サンプル問題を解いたときと同じように，自分で考えて解答を出したら，解説を読むようにしましょう。

(1) 練習問題　問 1

問 1　関数 searchMax は，要素数が 2 以上の配列 dataArray を引数として受け取り，要素の値の最大値を戻り値として返す。プログラム中の a，b に入れる字句の適切な組合せはどれか。ここで，配列の要素番号は 1 から始まる。

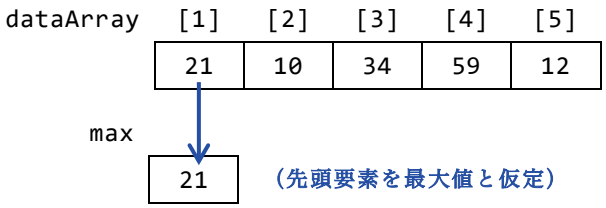
〔プログラム〕

```
○整数型: searchMax(整数型の配列: dataArray)    /* 関数の宣言 */
  整数型: max
  整数型: i
  max ← dataArray[1]    /* 配列の先頭要素を max として初期化する */
  for (i を 2 から dataArray の要素数まで 1 ずつ増やす)
    if ( a )
      b
    endif
  endfor
  return max
```

	a	b
ア	dataArray[i] ≦ max	max ← dataArray[i]
イ	dataArray[i] ≦ max	dataArray[i] ← max
ウ	dataArray[i] ＞ max	max ← dataArray[i]
エ	dataArray[i] ＞ max	dataArray[i] ← max

(問 1 の解説)

配列内の最大値を求めるプログラムです。まず関数 searchMax の引数の型と名称を確認すると，整数型の配列 dataArray で，この配列要素の中の最大値を求めて戻り値を整数型の max として返す処理であることを把握します。次に要素数が 5 の dataArray の例として，次のような図をメモすると処理が分かりやすくなります。



ここで，最初に先頭要素の dataArray[1]を最大値 max と仮定して，配列の 2 番目の要素以降と max の値を比較していく処理になっていることが，for 文の制御記述から分かります。

要素番号を i として，i を 2 から配列 dataArray の要素数まで 1 ずつ増やしながら，最大値 max と dataArray[i]を順番に比較していき，dataArray[i]が max より大きいときに，max の値を dataArray[i]の値で更新する処理になります。

したがって，空欄 a は dataArray[i] ＞ max，空欄 b は max ← dataArray[i]となるので（ウ）が正解です。

i の値の変化によって，先の dataArray の例で max が変わる様子を示すと次のようになり，最大値 max の 59 が戻り値として返されます。

i	比較前の max	dataArray[i]	比較後の max
（始め）	21		
2	21	10	21
3	21	34	34
4	34	59	59
5	59	12	59

正解　ウ

(2) 練習問題　問 2

問２　手続 replaceChar は，要素数が 1 以上の文字型の配列 charArray を引数として受け取り，先頭の要素から順に別の文字型の配列 repArray に格納していく。このとき配列 charArray の要素が空白文字（" "）であればアンダーバー（"_"）に置き換えて，配列 repArray に格納する。プログラム中の a，b に入れる字句の適切な組合せはどれか。ここで，配列の要素番号は 1 から始まる。

〔プログラム〕

```
○replaceChar(文字列型の配列: charArray)          /* 手続の宣言 */
  整数型: p ← 1                  /* 配列の要素位置を示す p を初期化する */
  文字列型の配列: repArray
  while ( a )
    if (charArray[p]が空白" "である)
      repArray[p] ← "_"
    else
      repArray[p] ← charArray[p]
    endif
    b
  endwhile
```

	a	b
ア	p が charArray の要素数以下	p ← p ＋ 1
イ	p が charArray の要素数以下	p ← p － 1
ウ	p が charArray の要素数より大きい	p ← p ＋ 1
エ	p が charArray の要素数より大きい	p ← p － 1

（問 2 の解説）

引数で渡された文字列型の配列要素 charArray を別の文字列型の配列 repArray に格納する処理です。配列要素 charArray が空白文字（" "）のときは，アンダーバー（"_"）を配列 repArray に格納しますが，この処理は while 文の処理の中の if 文で記述されています。

格納する位置を示す要素番号 p は整数型で，1 で初期化しています。この要素番号 p は配列の要素数まで 1 ずつ増やしながら処理を繰り返していきます。このことから，空欄 a

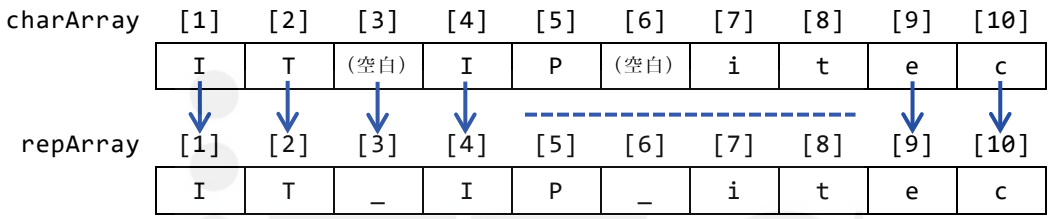
の条件式は(p が charArray の要素数以下)となることが分かります。

次に，endif の次の空欄 b ですがプログラム中に p の値に 1 を足す処理がないので，この位置で p ← p ＋ 1 の代入処理を実行します。以上から（ア）が正解になります。

この問題では while 文を使いましたが，繰返し処理が要素番号を変えていくだけの処理なので，for 文の制御記述を使って書いた方が p に 1 を足す処理を省略できるので簡潔になります。

注意事項として，while 文で指定する条件式は「処理を繰り返す条件」です。選択肢(ウ)，（エ）の（p が charArray の要素数より大きい）は処理を終了させるときの条件になっていて，繰り返す条件を否定した逆の意味になる条件なので注意してください。

このプログラムで，要素数が 10 の配列 charArray の要素を，先頭から配列 repArray に格納していくイメージを示すと次のようになります。



正解　ア

5　（付録）擬似言語の記述形式（IT パスポート試験用）

アルゴリズムを表現するための擬似的なプログラム言語（擬似言語）を使用した問題では，各問題文中に注記がない限り，次の記述形式が適用されているものとする。

〔擬似言語の記述形式〕

記述形式	説明
○手続名又は関数名	手続又は関数を宣言する。
型名：***変数名***	変数を宣言する。
/* ***注釈*** */	注釈を記述する。
// ***注釈***	
変数名 ← ***式***	変数に***式***の値を代入する。
手続名又は関数名(***引数***，…)	手続又は関数を呼び出し，***引数***を受け渡す。
if (***条件式 1***) 　***処理 1*** elseif (***条件式 2***) 　***処理 2*** elseif (***条件式 n***) 　***処理 n*** else 　***処理 n ＋ 1*** endif	選択処理を示す。 ***条件式***を上から評価し，最初に真になった***条件式***に対応する***処理***を実行する。以降の***条件式***は評価せず，対応する***処理***も実行しない。どの***条件式***も真にならないときは，***処理 n ＋ 1*** を実行する。 各***処理***は，0 以上の文の集まりである。 elseifと***処理***の組みは，複数記述することがあり，省略することもある。 else と***処理n ＋ 1*** の組みは一つだけ記述し，省略することもある。
while (***条件式***) 　***処理*** endwhile	前判定繰返し処理を示す。 ***条件式***が真の間，***処理***を繰返し実行する。 ***処理***は，0 以上の文の集まりである。
do 　***処理*** while (***条件式***)	後判定繰返し処理を示す。 ***処理***を実行し，***条件式***が真の間，***処理***を繰返し実行する。 ***処理***は，0 以上の文の集まりである。
for (***制御記述***) 　***処理*** endfor	繰返し処理を示す。 ***制御記述***の内容に基づいて，***処理***を繰返し実行する。 ***処理***は，0 以上の文の集まりである。

〔演算子と優先順位〕

演算子の種類		演算子	優先度
式		()	高
単項演算子		not ＋ －	↑
二項演算子	乗除	mod × ÷	↑
	加減	＋ －	↑
	関係	≠ ≦ ≧ ＜ ＝ ＞	↑
	論理積	and	↑
	論理和	or	低

注記　演算子 mod は，剰余算を表す。

〔論理型の定数〕

true，false

〔配列〕

一次元配列において“{”は配列の内容の始まりを，“}”は配列の内容の終わりを表し，配列の要素は，“[”と“]”の間にアクセス対象要素の要素番号を指定することでアクセスする。

（例）　要素番号が 1 から始まる配列 exampleArray の要素が{11，12，13，14，15}のとき，要素番号 4 の要素の値（14）は exampleArray[4]でアクセスできる。

二次元配列において，内側の“{”と“}”に囲まれた部分は，1 行分の内容を表し，要素番号は，行番号，列番号の順に“,”で区切って指定する。

（例）　要素番号が 1 から始まる二次元配列 exampleArray の要素が
{{11，12，13，14，15}，{21，22，23，24，25}}のとき，2 行目 5 列目の要素の値（25）は，exampleArray[2，5]でアクセスできる。

「IT パスポート試験 シラバス」Ver.6.0 対応 擬似言語補足資料

制作・編集　　アイテック　IT 人材教育研究部
発行日　2022 年 1 月 5 日　　第 1 版　第 1 刷
発行人　土元　克則
発行所　株式会社アイテック
〒143-0006　東京都大田区平和島 6-1-1　センタービル
電話　TEL：03-6877-6312 (教育事業本部)
http://www.itec.co.jp/