出題範囲とシラバスの一部改訂（ｉパス 6.0）の情報

ITパスポート試験（ｉパス）の出題範囲とシラバスの改訂が、IPA（独立行政法人情報処理推進機構）より発表されています*。

ここでは、改訂のポイントと試験対策、追加用語などについて解説します。

【改訂のポイント】

◆改訂の目的と適用時期

今回の改訂の目的は、高等学校の共通必履修科目に「情報Ｉ」が新設されることを踏まえ、IT パスポート試験の出題の見直しを実施し、高等学校等における試験の活用を促すこと、とされています。

新シラバスに沿った出題は2022年４月試験から適用されます。なお、試験時間や出題数、採点方式、合格基準は従来どおりであり、変更はありません。

◆改訂の内容

既存の出題内容に加えて、プログラミング的思考力やデータ利活用のための技術知識、情報倫理などの考え方を問う出題が強化されます。

具体的には、アルゴリズムとプログラム言語の分野、ヒューマンインタフェースを含む情報デザインの分野、業務分析・データ利活用の分野に関して、新しい用語がシラバスに追加されています。

また、プログラミング的思考の応用力を問うために、新たに「擬似言語」を用いた出題形式が登場します。

＊：「ITパスポート試験における出題範囲・シラバスの一部改訂について（高等学校情報科『情報Ｉ』への対応など）」
https://www.jitec.ipa.go.jp/1_00topic/topic_20211008.html

【試験対策】

出題内容が大幅に変更されたわけではなく、すでに過去の試験でも出題された用語が後からシラバスに追加されたケースも含まれます。そのため、まずはこれまでの教材の範囲と過去問題をしっかり対策しておくことが重要です。

ただし、今回追加された用語や項目は、今後、出題頻度が高まることが考えられますので、用語や項目の意味は一通り理解しておきましょう。

また、「疑似言語」については「アルゴリズム」をプログラミングに応用する方法を学ぶ新たな分野であり、受験の際に、限られた試験時間内にプログラムを読み解いて解答するには予習が必須となります。本資料を利用して、しっかり対策しておきましょう。

【追加された主な用語・項目】

以下に、情報デザインの分野と、データ利活用の分野に関して今回追加された用語・項目の中から、主なものを取り上げて解説します。

◆情報デザイン

デザインの原則（近接/整列/反復/対比）

製品や掲示物などをデザインする際、文字や絵といった情報を、見る人にできるだけわかりやすく印象的に見えるよう配置するのに有効とされる 4 つの原則です。

近　接	関連する情報どうしは近い位置にまとめ、グループ化すること。
整　列	関連する情報の間隔を整えて均等に配置すること。
反　復	同じ特徴のデザインを繰り返すことで一貫性をもたせること。
対　比	異なる要素には変化をつけて、違いを際立たせること。

シグニファイア

製品を人に使ってもらうためには、それが何にどのように役立つものかを理解してもらう必要があります。製品などのモノと人との間に関係性を築くための概念として、「アフォーダンス」と「シグニファイア」という考え方があります。

「アフォーダンス」とは、そのモノが何にどう役立つのか、という情報です。モノのもつアフォーダンスを人にうまく伝えるための要素を「シグニファイア」といいます。シグニファイアは、人が適切にモノを使えるように誘導するヒントとなるようなデザインです。

例として、Web ページ上で、そこにリンクが張られているという意味で青字の下線付き表記を使ったり、選択肢のうち選択が不可の状態であるものをグレーの文字で表記したりする、などがあります。

構造化シナリオ法

製品やサービスの仕様やデザインを決定するために、それとユーザとの接点を描いたシナリオを書いて段階的に仕様を検討する手法です。

シナリオは「価値のシナリオ」「行動のシナリオ」「操作のシナリオ」という 3 段階に分け

て作成します。最初に、製品のコンセプトとしての「価値のシナリオ」を作成し、そこから利用シーンごとの「行動のシナリオ」を作成し、さらに具体的な「操作のシナリオ」へと展開していきます。

価値のシナリオ	ユーザが何をしたいと思っているか、どんな状況でそう思うのか。
行動のシナリオ	ユーザがどんな行動を取れれば目標を達成できるのか。
操作のシナリオ	目標達成のためにユーザがとる具体的なふるまい。

インフォグラフィックス

情報やデータを、図を用いてわかりやすく表現することです。地図やグラフ、イラストや絵文字（ピクトグラム）などの例があります。

情報やデータを視覚的に表現したい場面で用いられます。文字に頼らないため、短時間で情報を見て取れる、言語や年齢に左右されずに情報を伝えやすい、というメリットがあります。

◆データ利活用

GIS データ

GIS は Geographic Information System（地理情報システム）の略で、地形や気候、自然、人口といった、地理と結びつく情報を扱うシステムです。地理的な位置情報とそれに結びつく情報（GIS データ）とを IT を用いて総合的に管理・解析し、さまざまな分析や意思決定に用いることができます。

GIS データは災害対策や都市計画、物流管理などの分野で利用されています。

クロスセクションデータ

ある特定の時刻・時点に、さまざまな場所や地域、グループで横断面的に記録されたデータのことです。データどうしを比較・照合することで、その時点での対象の相関関係を明らかにすることができます。

クロスセクションデータに対し、時間の経過に沿って記録された時系列データのことをタイムシリーズデータといいます。また、あるグループのクロスセクションデータを一定の間隔で繰り返し記録するなど、場所と時間の両方の要素をもつデータを、パネルデータといいます。

仮説検定/有意水準/第 1 種の誤り/第 2 種の誤り

仮説検定とは、ある仮説が統計学的に成り立つかどうかを、確率によって検討する方法です。想定している結論とは逆の仮説（帰無仮説）を立て、標本データを集めて仮説が成り立つ確率を求めることで、結論を受け入れるか棄却するかが決まります。その仮説が正しいとはいえないことを証明できれば、想定した結論（対立仮説）が正しいと判断できます。

たとえば、「データには〇がよくある」と想定した場合、「〇はめったにない」という帰無仮説を立てます。ここで、「めったにない」と判断する基準に使う確率を「有意水準」といいます。有意水準には、一般的に 5%や 1%などの値を用います。もし「〇」が有意水準を超えてデータに存在するなら、帰無仮説は棄却され、その対立仮説である「〇がよくある」が正しいことになります。

仮説検定を有効にするためには、無作為抽出された十分な量の標本データが必要です。また、適切な有意水準を選ぶことも重要です。標本データの不足や不適切な有意水準の設定は、判断の誤りにつながります。

帰無仮説が正しいのに帰無仮説を棄却することを、第 1 種の誤りといいます。有意水準が高いと、第 1 種の誤りが起きる確率も高くなります。

対立仮説が正しいのに帰無仮説を棄却しないことを、第 1 種の誤りといいます。標本データが少ないとデータの分散が大きくなり、第 2 種の誤りが起きやすくなります。

IT パスポート合格テキスト　2022 年版

新シラバス対応補足説明

IT パスポート試験では、プログラム言語の基本機能を模した架空の言語である「疑似言語」を使った問題が出題されます。問題文中に注記がない限り、次の記述形式が適用されます。「疑似言語の記述形式」は試験実施時の画面上でも参照できます。

【疑似言語の記述方法】

◆疑似言語の記述形式

処理のアルゴリズムは、流れ図などで表現すると、プログラム言語に置き換えて表しやすくなります。ここでは、アルゴリズムをプログラムに置き換えて記述する際の考え方を、次の疑似言語の記述形式に沿って説明します。

疑似言語の記述形式

○手続名または関数名	手続きまたは関数の名前を宣言する。
型名：変数名	使用する変数を宣言する。
/* 注釈 */ // 注釈	注釈を記述する。処理の説明などを自由に記述できる。
変数名 ← 式	変数に式の値を代入する。
手続名または関数名（引数，…）	手続きまたは関数を呼び出し、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。
	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 1 つだけ記述し、省略することもある。

while（条件式） 　処理 endwhile	前判定繰り返し処理を示す。 条件式が真の間、処理を繰り返し実行する。

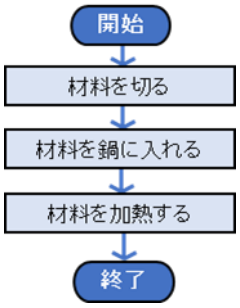
do 　処理 while（条件式）	後判定繰り返し処理を示す。 処理を実行し、条件式が真の間、処理を繰り返し実行する。
for（制御記述）※ 　処理 endfor	繰り返し処理を示す。 制御記述の内容に基づいて、処理を繰り返し実行する。

※for の制御記述には、繰り返す回数などを指定できる。

※if、while、do、for の構文にある「処理」の部分には、何らかの命令文が少なくとも１つ入る。

次の例は、３つの処理からなる順次構造のアルゴリズムに「cooking」という手続名をつけて疑似言語の形式で記述したものです。

最初に、手続名を「〇」に続けて宣言します。その後に処理を上から順番に記述します。「/**/」の中には、他の人が読んだときにわかりやすいよう、説明書きを入れることができます。



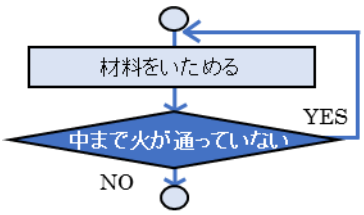
記述例

```
〇cooking          /*手続きの宣言*/
 材料を切る          /*処理 1*/
 材料を鍋に入れる    /*処理2*/
 材料を加熱する      /*処理3*/
```

◆繰り返し処理

繰り返し処理を表すには、while や for、do の構文を使います。いずれも条件式が真の間は処理を繰り返し実行し続け、条件式が偽となったときに繰り返しを終えます。

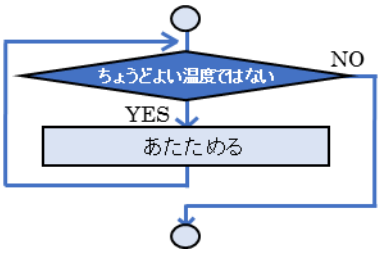
後判定型の繰り返し処理



記述例

```
〇itameru                    /*手続きの宣言*/
 do                          /*後判定繰り返し*/
   材料をいためる             /*処理*/
 While(中まで火が通っていない)  /*繰り返しの条件*/
```

前判定型の繰り返し処理



記述例

```
〇atatameru                          /*手続きの宣言*/
 while（ちょうどよい温度ではない）   /*前判定繰り返し（条件）*/
   あたためる                        /*処理*/
 endwhile                            /*繰り返し処理の終わり*/
```

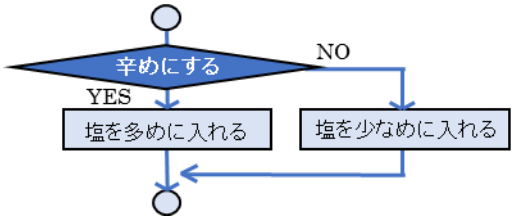
前述の後判定型の処理を、前判定型を使って次のように記述することもできます。

記述例

```
〇itameru
材料をいためる
while（中まで火が通っていない）
  材料をいためる
endwhile
```

◆分岐処理

条件の真偽によって異なる処理を選んで実行させるには、if の構文を使います。条件式が真の場合と偽の場合との、２通りの処理から選択できます。



記述例

```
〇sioaji                    /*手続きの宣言*/
 If(辛めにする)             /*分岐(条件)*/
   塩を多めに入れる         /*真の場合の処理*/
 else                       /*偽であれば*/
   塩を少なめに入れる       /*偽の場合の処理*/
 endif                      /*分岐処理の終わり*/
```

elseif を使うと３通り以上の分岐を作ることができます。

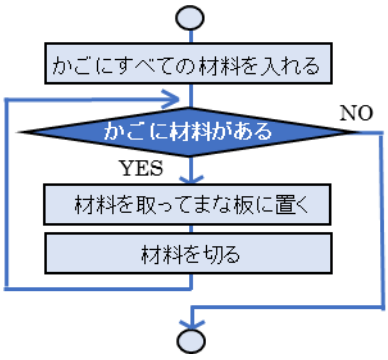
記述例

```
if(2倍辛くする)
   塩を2倍入れる
elseif(3倍辛くする)
   塩を3倍入れる
else
  塩を規定量入れる
endif
```

◆変数と配列の宣言

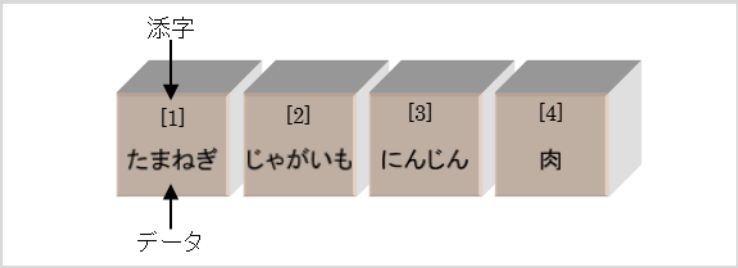
プログラムの中で文字や数値のデータを処理する場合、そのデータを格納するための変数を用意します。

疑似言語では、プログラムで使う変数や配列を最初に宣言する必要がありますが、流れ図では必要ありません。



前述の例の「かご」「まな板」に当たるような、処理中のデータを仮置きするための場所を変数といいます。変数には、格納するデータの種類に応じて、文字型、数値型（整数/実数）、論理型などがあります（データ構造については、テキスト 7-4「データ構造」を参照してください）。

複数のデータをまとまりとして扱うときには配列が利用できます。配列の要素は添字（インデックス）で区別されます。配列の添字には先頭が「１」からの場合と「０」からの場合があります。問題文中の指定に注意してください。



配列に含まれる要素は“ { } ”で囲んだ中に“, ”で区切りながら順番に記述します。配列に要素を追加したり、取り出したりするときは、添字を“ [ ] ”で囲んで指定することで、何番目の要素かを指定できます。

たとえば、要素番号が 1 から始まる配列「kago」の要素が｛たまねぎ , じゃがいも , にんじん , 肉｝のとき、kago［3］と指定すると、添字番号 3 の要素の値「にんじん」にアクセスできます。

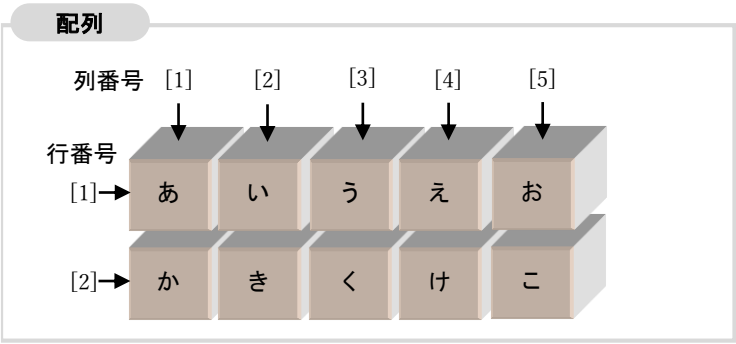
プログラム中で変数を使うときは、プログラムの最初に変数名とそのデータ型を「データ型：変数名」と宣言しておきます。

配列を使用するときは、最初に配列名とそのデータ型を「データ型：配列名」と宣言しておきます。

記述例

```
○junbi                                    /*手続きの宣言*/
  文字列型：manaita                        /*変数の宣言*/
  文字列型：kago = {たまねぎ, じゃがいも, にんじん, 肉}
                                           /*配列の宣言*/
                ⋮
```

要素を一列に並べて格納する配列を一次元配列といい、要素を行と列に区切って格納する配列を二次元配列といいます。



二次元配列を宣言するときは、要素の“｛　｝”の中でさらに“｛　｝”で囲み、そこに１行分の要素を“，”で区切って表します。続いて“，”で区切ったのちに２行目の“｛　｝”、３行目の“｛　｝”、…のように表します。

記述例

文字列型：hiragana = ｛｛あ,い,う,え,お｝,｛か,き,く,け,こ｝｝

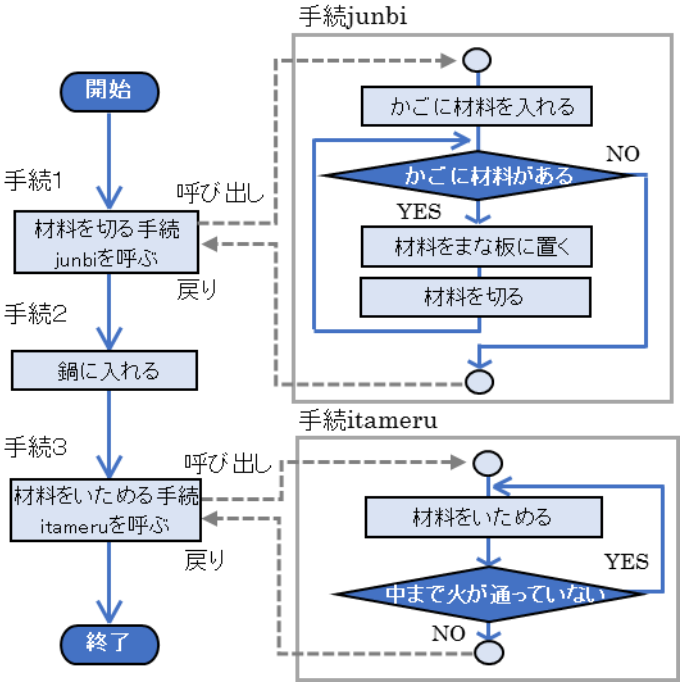
要素番号は、“［　］”内に行番号、列番号の順に“，”で区切って指定します。たとえば、上記の配列に hiragana[2,4]と指定すると、２行目の４番目の列にある要素「け」にアクセスできます。

配列の宣言では、｛　｝に要素を並べる代わりに［　］に要素数だけ宣言することもあります。

◆手続き（関数）の呼び出し

ある目的のためのひとまとまりの処理に名前をつけて、手続き（関数）として宣言しておくと、ほかの手続きの中で、その関数を部品として呼び出して使うことができます。

C 言語などでは部品となる手続きのことを関数と呼ぶことがあります。関数を部品として利用するとコーディングの手間が省ける上、プログラム全体の構造がわかりやすくなり、保守性が向上します。



関数を呼び出すとき、処理に使うためのデータを関数に引き渡すことがあります。これを引数といいます。引数の必要な関数を呼び出すには、「関数名（引数,・・・）」と宣言します。引数は配列であってもかまいません。

記述例

```
mojicount1（あ,い,う,え,お）
        /*引数に「あいうえお」を使う関数 mojicount1 の宣言*/
mojicount2（文字列型の配列：hiragana）
        /*引数に配列「hiragana」を使う関数 mojicount2 の宣言*/
```

また、呼び出した関数から処理結果のデータを返してもらう場合もあります。これを戻り値といいます。戻り値のある関数を使用するには、「戻り値のデータ型：関数名（引数）」と宣言します。

記述例

```
実数型：mojicount3（文字列型の配列：hiragana）
        /*引数に配列を使い実数値を返す関数の宣言*/
```

◆疑似言語で使う演算子

条件式や処理で比較や計算をする場合に使う演算子には次のようなものがあります。

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

※演算子 mod は剰余算を表す。

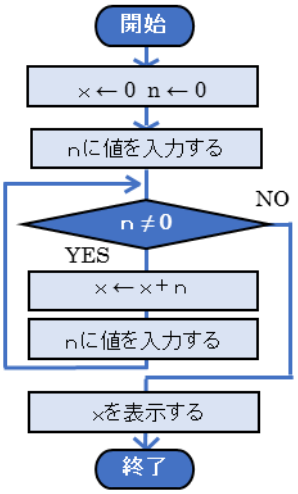
◆合計を計算するプログラム

ここでは、キーボードで数字を１つずつ入力し、その合計を計算して表示するアルゴリズムを疑似言語で記述した例を示します。「0」が入力されたら、計算の処理を終え、結果を表示します。

プログラムの構造は、大きく処理の準備として最初に一度だけ行う（初期処理）部分と、プログラムの主たる目的を果たす（主処理）部分とに分けられます。

通常、初期処理では手続き名の宣言と手続き中に使う関数や変数の宣言、初期値の設定などを行います。ここでは、合計の値を格納するための変数 x と、キーボードから入力された値を格納するための変数 n を宣言します。それぞれの変数には初期値として「0」を格納しておきます。

続いて合計の処理を始めます。キーボードから入力された値を n に格納し、x 内の合計値に n を合計したものを x に格納する、という処理を繰り返し、「0」が入力されたら x の値を表示して終了します。



記述例

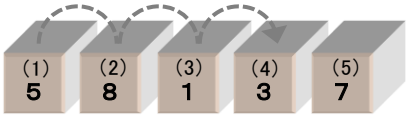
```
○goukei                 /*合計を求める手続き goukei*/
  整数型: x , n         /*x は合計、n は入力値*/
  x ← 0                 /*変数 x に初期値 0 をセット*/
  n ← 0                 /*変数 n に初期値 0 をセット*/
  n に値を入力する      /*キーボードから入力*/
  while(n ≠ 0)          /*nが 0 以外なら以下を繰り返す*/
    x ← x+n             /*x と n の和をxに格納*/
    n に値を入力する    /*キーボードから入力*/
  endwhile              /*ここまでを繰り返す*/
  x を表示する          /*xにある合計値を表示する*/
```

疑似言語の問題では、前提としてこのような関数を用いる、と問題文中に指示されていることがあります。たとえば、「キーボードから入力された値を受け取る関数として input () という関数が用意されており、値を画面に表示する関数として output () という関数が用意されている」という前提があるとしたら、「n に値を入力する」は「n ← input ()」と表記でき、「x を表示する」は「output（x)」と表記できることになります。

疑似言語は架空の言語なので記述に厳格なルールはありません。問題文にそのつど説明があるので、指示に従って解答してください。

◆探索するプログラム

ここでは、いくつかの数値の中から、目的の数値を探し出すアルゴリズムを疑似言語で記述した例を示します。配列 n に 5 つの異なる数値が入っており、その先頭から順に「3」を探し、見つかれば何番目であるかを表示します。見つからなければ「なし」と表示します。配列 n の添字は 1 から始まるとします。



「3」が何番目に
あるか探索する

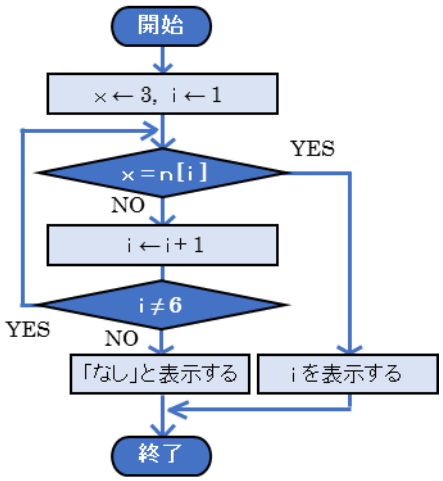
まず初期処理として、探す値（ここでは 3）を格納するための変数 x と、配列の添字を格納するための変数 i を宣言します。変数 x には初期値として「3」を入れます。変数 i には初期値として「1」を入れておきます。

続いて探索の処理を始めます。配列 n の値を先頭から順に x と比較し、合致すればその添字番号を表示して終了します。合致しなければ次の値を x と比較する処理を繰り返します。

もし配列内の 5 つの値がすべて x と合致しなかったら「なし」と表示して終了します。

記述例

```
○tansaku(整数型の配列:n)
                    /*配列から値を探索する手続き tansaku*/
整数型: x , i       /*x は探索値、i は添字*/
x ← 3               /*変数 x に探索値3をセット*/
i ← 1               /*変数 i に初期値 1 をセット*/
do                  /*do～while の繰り返し開始*/
  if(x=n[i])        /*分岐(条件) */
   i を表示する     /*x=n[i]が真の場合の処理*/
   return           /*処理を終えて出る*/
  else              /*x=n[i]が偽であれば*/
   i ← i+1          /*偽の場合の処理*/
  endif             /*偽の場合の処理の終わり*/
while(i≠6)          /* i≠6 の間ここまでを繰り返し*/
「なし」と表示する  /* i=6 の場合の処理*/
```



記述例の 9 行目で使われている「return」は、それ以降の処理をやめて手続きを終了するための命令文です。このような、「疑似言語の記述形式」で取り上げられていない命令文が問題文中に使われる場合もあります。問題文中に説明があれば参照してください。

【疑似言語の例題】

新たに出題されるプログラミング的思考力を問う擬似言語については、IPA（情報処理推進機構）よりサンプル問題が公開されています。ここではサンプル問題を基に擬似言語の解き方を解説していきます。

◆文字を出力するプログラム

「☆」「★」を出力するプログラム「printStars」の問題について考えてみましょう。

手続き「printStars」は、「☆」と「★」を交互に、引数「num」で指定された数だけ出力します。たとえば、num＝5 のときは「☆★☆★☆」と出力します。引数「num」の値が 0 以下のときは何も出力しません。

このときプログラム内の空欄 a と b にどのような記述を入れればよいかを、以下の選択肢から選びます。

aの選択肢	bの選択肢
do	while（cnt が num 以下）
while（cnt が num 以下）	while（cnt が num より小さい）
while（cnt が num より小さい）	endwhile

```
［プログラム］
○printStars（整数型：num）        /*手続きの宣言*/
  整数型：cnt ← 0                  /*出力数を初期化*/
  文字列型：starColor ← ”SC1”   /*最初は☆を出力*/
  ┌──────┐
  │   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   │
  └──────┘
```

printStars には引数として整数型の「num」が渡されます。何文字出力したかをカウントするための変数「cnt」と、色を区別するための変数「starColor」を使っています。

「starColor」には「☆」を示す「SC1」か、「★」を示す「SC2」が交互に格納されます。

◆プログラムの読み解き方

プログラム全体の構造を、処理の準備として最初に一度だけ行う部分（初期処理）と、プログラムの目的を果たす主たる処理の部分とに分けて考えます。

[プログラム]

```
〇printStars(整数型:num)          …手続きの宣言
  整数型:cnt ← 0                  …変数の宣言
  文字列型:starColor ← ”SC1”      …変数の宣言
```

上の部分が、手続きや変数の宣言と初期値の設定をしている初期処理です。主たる処理は以下の部分になります。

```
        a
  if(starColor が”SC1”と等しい)
    ”☆”を出力する
    starColor ← ”SC2”
  else                                  if～endif
    ”★”を出力する                       分岐処理
    starColor ← ”SC1”
  endif
  cnt ← cnt＋1                          …変数 cnt の処理
        b
```

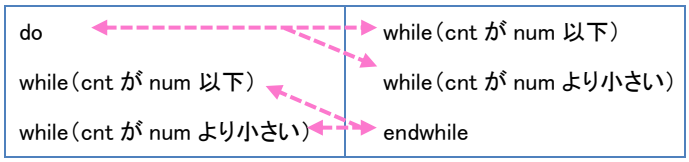
主たる処理の構造は、if～endif の構文による分岐処理と、変数「cnt」に 1 を加える処理とを、空欄 a と空欄 b で挟んだ形になっていることがわかります。

分岐処理では、starColor が“SC 1 ”なら「☆」を出力し、“SC2”なら「★」を出力するという、色分けしながら出力する処理を行っています。出力後には、次に反対の色を出力するために starColor の値を変更しています。分岐処理の後、出力した文字数を数える変数「cnt」に 1 をプラスします。

文字は引数「num」で指定された数だけ出力する必要がありますので、空欄 a と b を、「num」の数だけ文字が出力されるような繰り返しの構文になるように埋めればよいことがわかります。

◆選択肢の読み解き方

選択肢の内容から、空欄 a と b は繰り返しの構文である do～while の後判定型か、while～endwhile の前判定型という組合せになるよう、選択する必要があります。



※「～以上（≧）」、「～以下（≦）」という場合、＝も含むことに注意する。
※「～より小さい」、「～を超える」という場合は＝は含まれない。

「num」の数だけ文字を出力する繰り返しの構文にするには、出力される文字数が多すぎたり少なすぎたりしないよう注意します。

まず、引数「num」の値が 0 以下のときに何も出力せずに終わるためには、前判定の繰り返し構文が適しています。do～while の後判定型では、「num」が 0 以下だったときにも 1 文字出力してしまうため、適していません。

前判定型の while（cnt が num 以下）を使うと、「num」の数だけ文字を出力して cnt＝num の状態になったときにも条件はまだ真となり、さらにもう一度出力の処理を行ってしまうため、1 文字多く出力されてしまいます。

このため、while（cnt が num より小さい）～endwhile を用いるのが適していますので、空欄 a に while（cnt が num より小さい）、空欄 b に endwhile を入れます。