文部科学省後援

情報システム試験

令和７年度後期 情報検定

＜実施　令和8年2月8日（日）＞

プログラミングスキル

（説明時間　10：00～10：10）

（試験時間　10：10～11：40）

・試験問題は試験開始の合図があるまで開かないでください。

・解答用紙（マークシート）への必要事項の記入は，試験開始の合図と同時に行いますので，それまで伏せておいてください。

・試験開始の合図の後，次のページを開いてください。**＜受験上の注意＞**が記載されています。必ず目を通してから解答を始めてください。

・試験問題は，すべてマークシート方式です。正解と思われるものを１つ選び，解答欄の()をＨＢの黒鉛筆でぬりつぶしてください。２つ以上ぬりつぶすと，不正解になります。

・辞書，参考書類の使用および筆記用具の貸し借りは一切禁止です。

・電卓の使用が認められます。ただし，下記の機種については使用が認められません。

＜使用を認めない電卓＞

１．電池式（太陽電池を含む）以外の電卓

２．文字表示領域が複数行ある電卓（計算状態表示の一行は含まない）

３．プログラムを組み込む機能がある電卓

４．電卓が主たる機能ではないもの

＊パソコン（電子メール専用機等を含む），携帯電話，スマートフォン，タブレット，電子手帳，電子メモ，電子辞書，翻訳機能付き電卓，音声応答のある電卓，電卓付き腕時計，時計型ウェアラブル端末等

５．その他試験監督者が不適切と認めるもの

一般財団法人　職業教育・キャリア教育財団

＜受験上の注意＞

１．この試験問題は18ページあります。ページ数を確認してください。
乱丁等がある場合は，手をあげて試験監督者に合図してください。
※問題を読みやすくするために空白ページを設けている場合があります。

２．解答用紙（マークシート）に，受験者氏名・受験番号を記入し，受験番号下欄の数字をぬりつぶしてください。正しく記入されていない場合は，採点されませんので十分注意してください。

３．試験問題についての質問には，一切答えられません。自分で判断して解答してください。

４．試験中の筆記用具の貸し借りは一切禁止します。

５．試験を開始してから30分以内は途中退出できません。30分経過後退出する場合は，もう一度，受験番号・マーク・氏名が記載されているか確認して退出してください。なお，試験終了５分前の合図以降は退出できません。試験問題は各自お持ち帰りください。

６．試験後の合否結果（合否通知），および合格者への「合格証・認定証」はすべて，Ｗｅｂ認証で行います。
①情報検定（J検）Ｗｅｂサイト合否結果検索ページ及びモバイル合否検索サイト上で，ディジタル「合否通知」，ディジタル「合格証・認定証」が交付されます。
②団体宛には合否結果一覧ほか，試験結果資料一式を送付します。
③合否等の結果についての電話・手紙等でのお問い合わせには，一切応じられませんので，ご了承ください。

問題１　次のデータ構造に関する記述を読み，各設問に答えよ。

二分木とはデータ構造の一つであり，木構造を利用したデータの管理を行う。

二分木では各ノードの子は最大で２つまでであることから，左の子と右の子が明確に区別される。図１中の「○」をノード，ノード間を結ぶ線を枝という。枝で結ばれている２つのノードには上が親，下が子という親子関係が存在する。一番上位にあるノードを根，子を持たないノードを葉という。

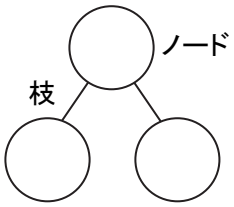

図1　二分木構造の例

＜設問１＞　次の二分木に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

図２の二分木を探索することを考える。

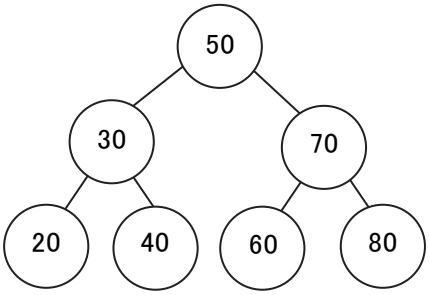

図2　二分木

この木に対して，さまざまな探索方法でノードを探索する順序を調べる。まず，幅優先探索の場合，根から順に，左から右へノードを探索する。このとき幅優先探索の順序は［　(1)　］となる。

次に，深さ優先探索について順序を調べる。深さ優先探索については，左から右まで外周をたどりながら，ノードを縦方向（深さ）に探索を行うが，さらに中間順と後行順の２種類を検討する。深さ優先探索（中間順）の場合，左部分木→根→右部分木の順に探索するため，順序は［　(2)　］となる。深さ優先探索（後行順）の場合，左部分木→右部分木→根の順に探索するため，順序は［　(3)　］となる。

(1) の解答群

ア．50 → 30 → 20 → 40 → 70 → 60 → 80
イ．50 → 30 → 70 → 20 → 40 → 60 → 80
ウ．50 → 70 → 30 → 20 → 40 → 60 → 80
エ．50 → 70 → 80 → 60 → 30 → 40 → 20

(2)，(3) の解答群

ア．20 → 30 → 40 → 50 → 60 → 70 → 80
イ．20 → 40 → 30 → 60 → 80 → 70 → 50
ウ．50 → 30 → 20 → 40 → 70 → 60 → 80
エ．50 → 70 → 80 → 60 → 30 → 40 → 20

＜設問２＞　次の二分探索木のデータ追加，削除に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

図２の二分木を初期状態としてノードの追加と削除を考える。ここでノードの値は node.value，ノードの左の子の値は node.left，ノードの右の子の値は node.right とし，追加または削除したい値を value と表現する。また，二分探索木のノードと子ノードの関係は node.left＜node.value＜node.right とする。

[追加の説明]

根から順番に追加する場所の探索を始めるため，最初は根が node となり，value ＜ node.value で比較をする。この条件が成り立つ場合，［ (4) ］が存在しなければ新しい node を追加して，値に value を設定する。この条件が成り立たない場合，［ (5) ］が存在しなければ新しい node を追加して，値に value を設定する。もし，子が存在して条件が成り立つ場合は node ← node.left を，子が存在して条件が成り立たない場合は node ← node.right を設定して繰り返す。

[削除の説明]

根から削除対象のノード探索を始める。［ (6) ］になった場合，削除処理を実行するが，次の場合が考えられる。

1. 子がない場合は node を削除する。
2. node.left が存在する場合，node ← node.left をして削除する。
3. node.right が存在する場合，node ← node.right をして削除する。

(4)，(5) の解答群

ア．node.left　　イ．node.right　　ウ．node.value

(6) の解答群

ア．value ＜ node.left　　イ．value ＜ node.right　　ウ．value = node.value

問題２　次の流れ図の説明を読み，流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

［流れ図の説明］

エラトステネスのふるいは，ある整数Ｎまでの素数を効率的に求めるためのアルゴリズムである。素数とは，1とその数自体でしか割り切れない2以上の整数である。

エラトステネスのふるいでは，2 から順に整数の倍数を削除していき，残った整数が素数となる。具体的な手順は以下の方法で求める。なお，配列の添字は 0 から始める。

手順１：0からＮまでの整数を「素数候補」として昇順に配列numへ格納する。
手順２：最小の素数候補ｉを取り出す。
手順３：ｉの倍数（2i，3i，4i，…）をすべて候補から削除（-1を格納）する。
手順４：次の候補ｉ＋1に進み，まだ削除されていなければ素数となる。
手順５：ｉがＮに達するまで手順3～4を繰り返し，残った数をすべて素数とする。

(1) の解答群

ア．num[i] ← 0　　イ．num[i] ← 1
ウ．num[i] ← i　　エ．num[j] ← i

(2) の解答群

ア．i ← 0　　イ．i ← 1
ウ．i ← 2　　エ．i ← N

(3) の解答群

ア．ｉがＮ以下である　　イ．ｉが偶数である
ウ．ｉが削除済みである　　エ．ｉが素数候補に残っている

(4) の解答群

ア．j ← 0　　イ．j ← i ＋ 1
ウ．j ← i － 1　　エ．j ← i × 2

(5) の解答群

ア．num[i] ← j　　イ．num[j] ← i
ウ．num[i] ← -1　　エ．num[j] ← -1

(6) の解答群

ア．i ← i ＋ 1　　イ．i ← i × 2
ウ．j ← j ＋ 1　　エ．j ← j × 2

［流れ図］

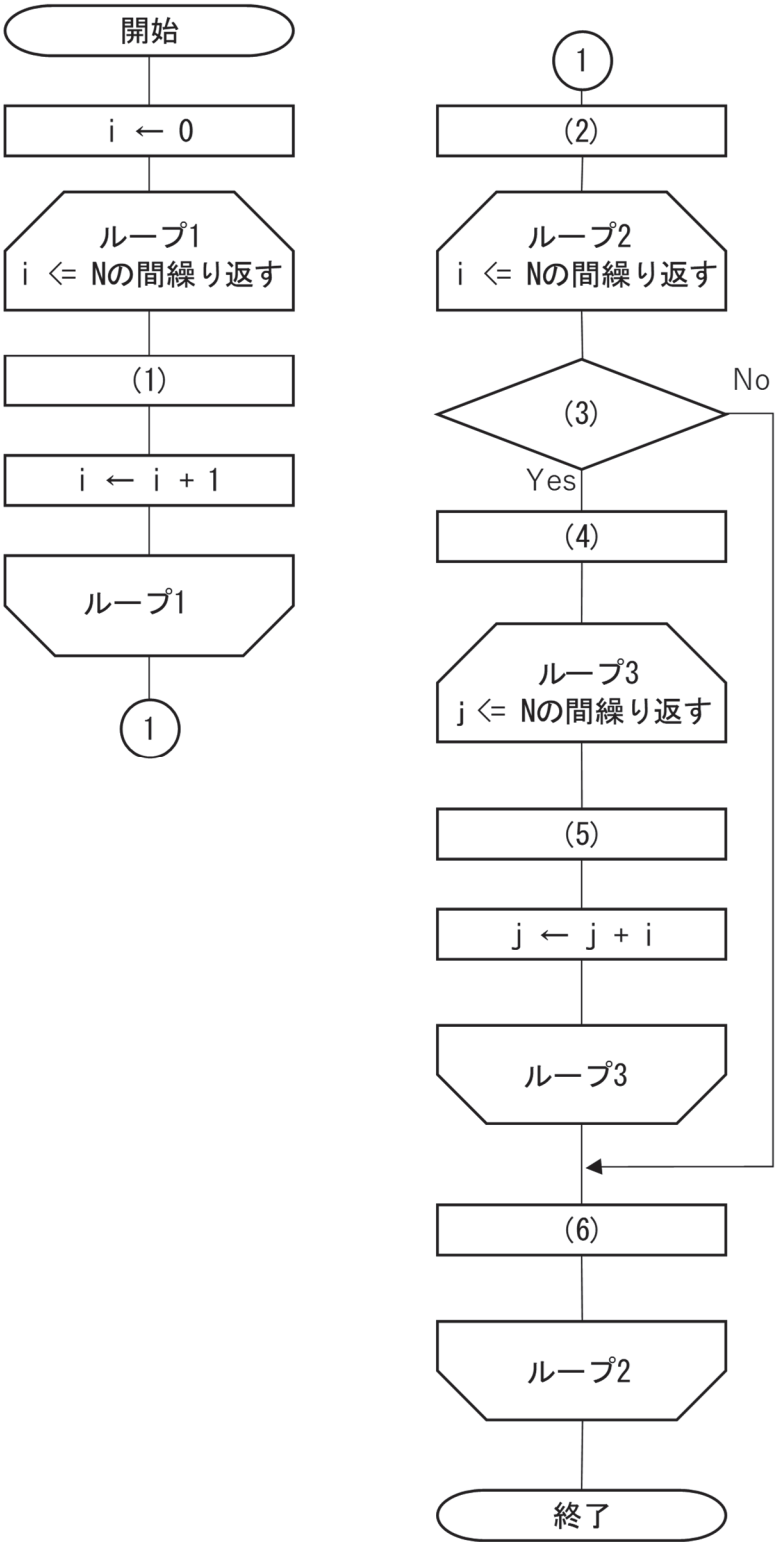
開始
i ← 0
ループ1
i <= Nの間繰り返す
(1)
i ← i + 1
ループ1
1
1
(2)
ループ2
i <= Nの間繰り返す
(3)
No
Yes
(4)
ループ3
j <= Nの間繰り返す
(5)
j ← j + i
ループ3
(6)
ループ2
終了

図　流れ図

問題３　次の配列処理に関する記述を読み，各設問に答えよ。なお，配列の添字は1から始まる。

［配列処理について］

図１のように配列の内容を反転する処理を考える。

反転前	100	200	300	400	500
	↓				
反転後	500	400	300	200	100

図１　反転の例

＜設問１＞　次の反転処理の記述を読み，流れ図の［　　］に入れるべき適切な字句を解答群から選べ。

［反転処理］

１次元配列を引数で受け取り，配列内の要素を反転した１次元配列 wk（大域変数）を返す rev である。１次元配列は配列 ary に，配列の要素数を size に受け取る。

［流れ図］

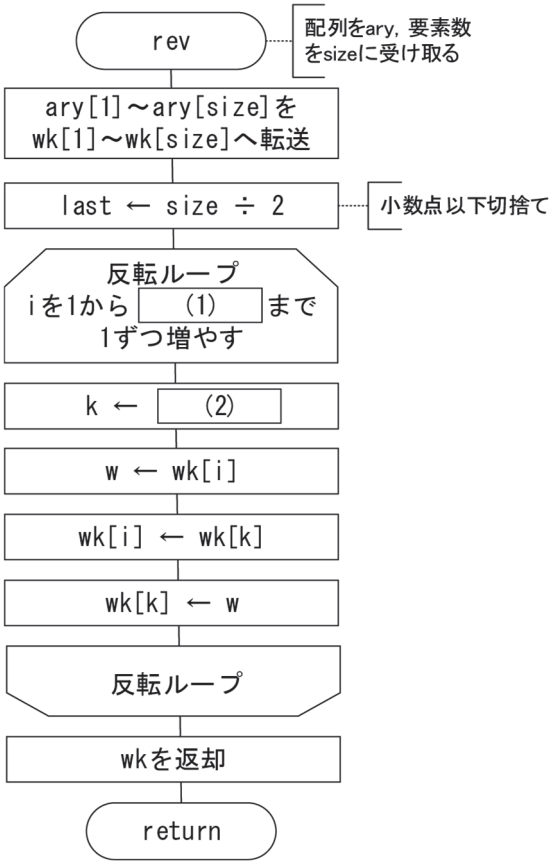

図２　配列を反転する流れ図

(1) の解答群

ア．last － 1　　イ．last

ウ．size　　エ．size － 1

(2) の解答群

ア．last － i　　イ．last － i ＋ 1

ウ．size － i　　エ．size － i ＋ 1

［２次元配列の反転処理について］

１次元配列で行った反転処理を２次元配列の行および列要素について行う。

図３は１行目だけ反転した例で，図４は１列目だけを反転した例である。

反転前

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

→

反転後

5	4	3	2	1
6	7	8	9	10
11	12	13	14	15

図３　１行目だけを反転した例

反転前

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

→

反転後

11	2	3	4	5
6	7	8	9	10
1	12	13	14	15

図４　１列目だけを反転した例

＜設問２＞　次の２次元配列の反転処理の記述を読み，流れ図の［　　］に入れるべき適切な字句を解答群から選べ。

［２次元配列の反転処理］

２次元配列の中で特定の１行を反転する rowRev と，特定の１列を反転する colRev である。特定の行または列は，それぞれ引数 row および col に受け取る。

なお，２次元配列は ary，配列の行数は ROWSIZE，列数は COLSIZE に格納されており，大域変数として定義されている。また，流れ図中で呼び出す rev は図２の流れ図である。

［流れ図］

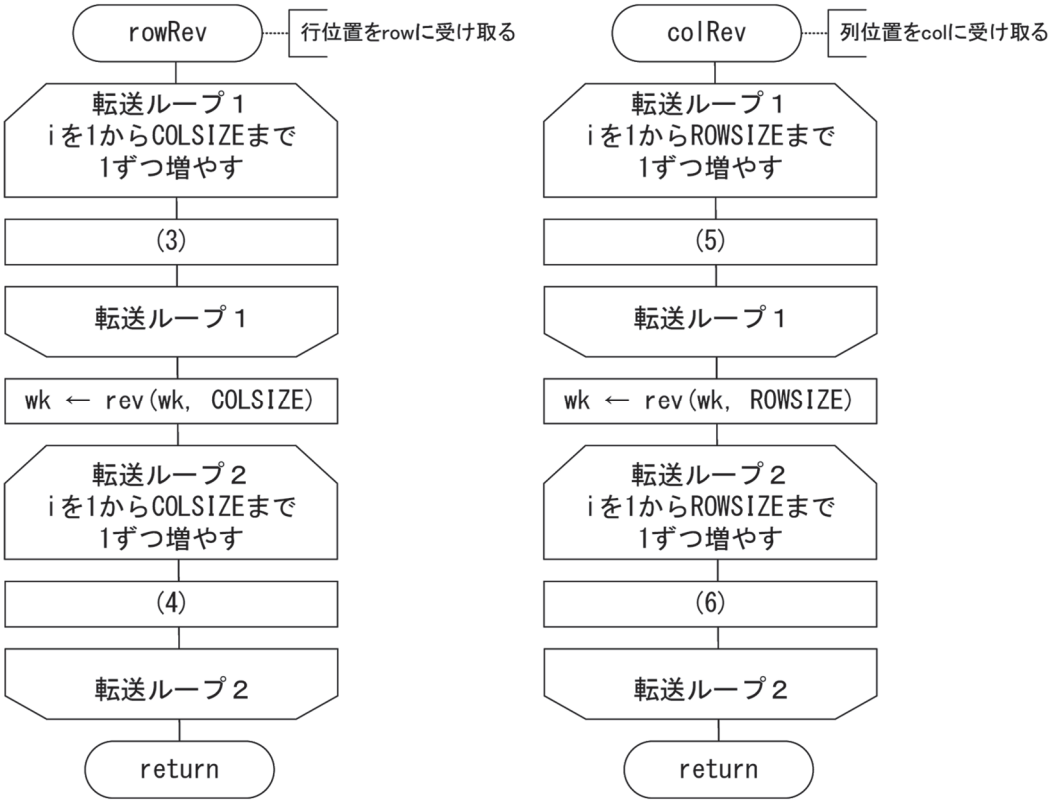

図５　rowRev と colRev の流れ図

(3)，(5) の解答群

ア．wk[i] ← ary[col][i]　　イ．wk[i] ← ary[i][row]

ウ．wk[i] ← ary[i][col]　　エ．wk[i] ← ary[row][i]

(4)，(6) の解答群

ア．ary[col][i] ← wk[i]　　イ．ary[i][col] ← wk[i]

ウ．ary[i][row] ← wk[i]　　エ．ary[row][i] ← wk[i]

＜設問３＞　配列 ary の最初の内容が次の図６のようであるときに，図２および図５の流れ図を使用して図７の流れ図を実行した後の配列 ary の内容を解答群から選べ。

［配列aryの最初の内容］

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

図６　配列 ary の最初の内容

［流れ図］

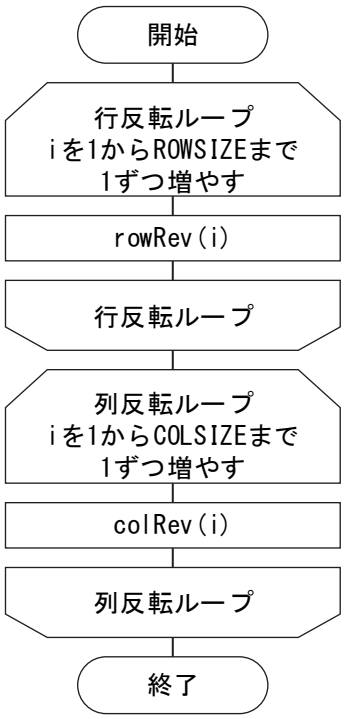

図７　rowRev，colRev，rev を呼び出す流れ図

(7) の解答群

ア.

15	14	13	12	11
10	9	8	7	6
5	4	3	2	1

イ.

11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

ウ.

1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

エ.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

問題４，問題５で使用する擬似言語の仕様は以下のとおりである。

［擬似言語の記述形式の説明］

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*	注釈を記述する。
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 . は，メンバ変数又はメソッドのアクセスを表す。
演算子 mod は，剰余算を表す。

〔論理型の定数〕

true，false

〔配列〕

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。なお，二次元配列の要素番号は，行番号，列番号の順に“,”で区切って指定する。

“{”は配列の内容の始まりを，“}”は配列の内容の終わりを表す。ただし，二次元配列において，内側の“{”と“}”に囲まれた部分は，１行分の内容を表す。

〔未定義，未定義の値〕

変数に値が格納されていない状態を，“未定義”という。変数に“未定義の値”を代入すると，その変数は未定義になる。

問題４　次のプログラムの説明を読み，各設問に答えよ。なお，配列の添字は0から始まる。

２次関数 F(x) と x＝a，x＝b，さらに x 軸の４つの線で囲まれる領域(図１の斜線の部分)の面積を求めるには，図２のような複数の長方形の面積の総和で近似するのが最も簡単である。しかし，本来は曲線であるところを長方形で近似しているため，はみ出たり欠けたりするので，誤差が大きくなることがある。この誤差を軽減するにはいろいろなアルゴリズムがある。

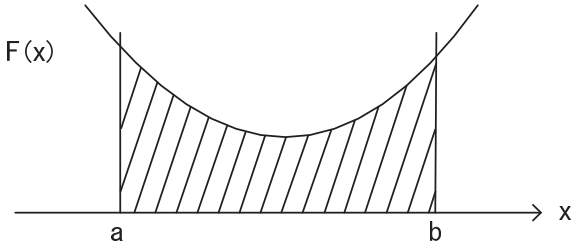

図１　求める面積

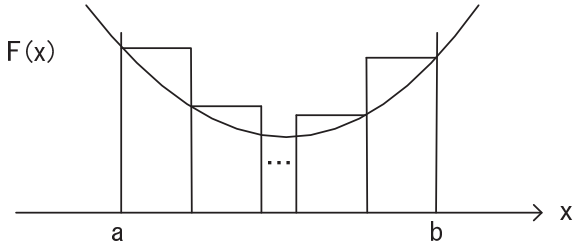

図２　長方形で近似するイメージ

＜設問１＞　次の台形則に関する説明を読み，プログラム中の［　　］に入れるべき適切な字句を解答群から選べ。

［プログラムの説明］

台形則とは，図２のように長方形で近似するところを図３のように台形で近似することにより，本来の面積との誤差を少なくする方法である。

ここで，２次関数は $F(x)=4x^2+3x+5$ とする。なお，台形の面積は，

(上底＋下底)÷２×高さ

で求めることができる。

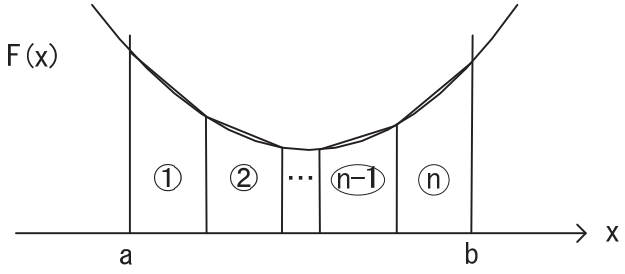

図３　台形で近似するイメージ

はじめに，面積を求める範囲(a～b)を x 軸に対して n 等分し高さ(h)とする。

図３での①にあたる１番左側に存在する台形の面積は，左側の縦線(上底)は x＝a，右側の縦線(下底)は x＝a＋h として求める。図４に，①の部分を 90° 右回転したイメージを示す。

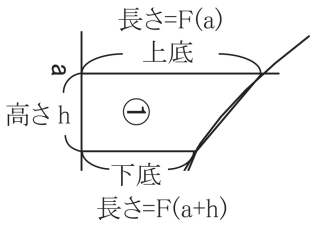

図４　①の部分を 90° 右回転したイメージ

①の面積は，上底の長さはF(a)，下底の長さはF(a＋h)で求めることができる。

図３での②にあたる左側から２番目に存在する台形の面積は，上底の長さは F(a＋h)，下底の長さはF(a＋2h)で求めることができる。

同様に，図３での(n)にあたる１番右側の台形の面積まで求めながら，面積の総和を求める。

［プログラム１］

```
○実数型: Quadratic_func(実数型: x)
  /* ２次関数 F(x) */
  return 4 × x × x + 3 × x + 5.0

○実数型: Trapezoid(実数型:a, 実数型:b, 実数型:n)
  整数型: i
  実数型: h, s, sum, x1, x2
  /* 合計値を初期化する */
  [   (1)   ]
  /* 面積を求める範囲を分割する */
  h ← [   (2)   ]
  /* 一つずつの台形の面積を求めながら累積する */
  for (i を 0 から i < n の間 1 ずつ増やす)
      x1 ← a + h × i
      x2 ← a + h × (i + 1)
      s ← (Quadratic_func(x1) + Quadratic_func(x2)) ÷ 2.0 × h
      [   (3)   ]
  endfor
  return sum
```

(1)，(3) の解答群

ア．s ← 0.0　　イ．sum ← 0.0

ウ．s ← s + sum　　エ．sum ← sum + s

(2) の解答群

ア. (a - b) ÷ (n - 1)　　イ. (a - b) ÷ n

ウ. (b - a) ÷ (n - 1)　　エ. (b - a) ÷ n

＜設問２＞　次のプログラムの変更に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。なお，(1)～(3)には設問１と同じ字句が入る。

台形則では，面積を求める範囲(a～b)を x 軸に対して分割する数をある程度まで増やしていくと精度が上がることが知られている。そこで，分割数を指定するプログラム１を変更し，分割数を１から１ずつ増やしながら，前回と今回の面積の総和との差が指定した数値(lim)より大きい間は分割数を１ずつ増やすことにした。

［プログラム２］

```
○実数型: Quadratic_func(実数型: x)
  /* ２次関数 F(x) */
  return 4 × x × x + 3 × x + 5.0

○実数型: Trapezoid(実数型:a, 実数型:b, 実数型:lim)
  整数型: i, n
  実数型: h, s, sum, x1, x2
  実数型の配列: tmp ← {} // 要素数 0 の配列

  tmp の末尾に 0.0 を追加する
  /* 分割数を増やしながら，面積を求める */
  n ← 1
  while (true)
      (1)
      /* 面積を求める範囲を分割する */
      h ← (2)
      for (i を 0 から i < n の間 1 ずつ増やす)
          x1 ← a + h × i
          x2 ← a + h × (i + 1)
          s ← (Quadratic_func(x1)+ Quadratic_func(x2)) ÷ 2.0 × h
          (3)
      endfor
      tmp の末尾に sum を追加する
```

```
    /* Math.abs は絶対値を求める関数 */
    if ( Math.abs(tmp[n-1] - tmp[n]) > lim )
        (4)
    else
        break
    endif
endwhile
return sum
```

(4) の解答群

ア. n ← n - 1　　イ. n ← n + 1

ウ. n ← n ÷ 2

問題５　次の二分探索法に関する記述を読み，各設問に答えよ。なお，擬似言語の仕様は問題４と同様である。

二分探索法(Binary Search)は，昇順または降順に並んだ一次元配列からデータを２分（Binary）しながら探索する方法である。単純に１つずつデータを比較する方法と比べ，探索範囲を２分の１に狭めながら比較できるため，高速な検索が期待できる。

［二分探索法によるプログラムの説明］

binarySearch()は，二分探索法により一次元配列中から任意の数値データを検索する関数である。昇順に整列された各数値データを格納する一次元配列 data[0]～data[n－1]，配列 data の大きさ n，探したいデータ（探索値）x，を順に引数として受け取り，探索値の見つかった添字（見つからない場合は－1）を返却する。なお，本問題における配列の要素番号は 0 から始める。

流れ図および binarySearch()の処理は次のとおりとする。

①　探索範囲の先頭要素の添字を low，末尾要素の添字を high，終了条件の管理フラグを sw とする。なお，初期値は，low＝0，high＝n－1，sw＝－1 である。

②　探索範囲の中央要素となる data[m]を x と比較する。なお，m＝(low＋high)÷2 とし，小数点以下は切り捨てる。次の手順を繰り返し，探索する。

Ⅰ data[m]＝x なら，探索値が見つかったので指定の処理を実行し終了する。

Ⅱ data[m]＜x なら，より大きな位値に探索値が存在するため low＝m＋1 とし，次の探索範囲を，配列の要素位置が m より大きい方とする。

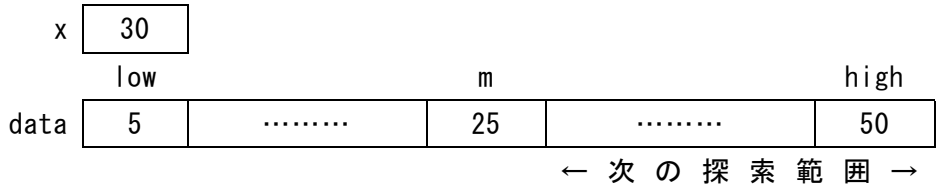

図１　比較例１

Ⅲ data[m]＞x なら，より小さな位値に探索値が存在するため high＝m－1 とし，次の探索範囲を，配列の要素位置が m より小さい方とする。

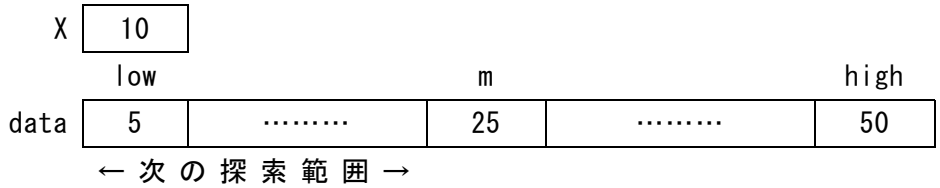

図２　比較例２

③　low＞high または data[m]＝x となるまで,②を繰り返す。low＞high の場合は，データ x が配列 data に存在しないことになる。

＜設問１＞　二分探索法による以下の流れ図およびプログラム中の［　　］に入れるべき適切な字句を解答群から選べ。なお，以下の流れ図およびプログラムは同じ処理を表しており，除算は小数点以下を切り捨てる。

［流れ図］

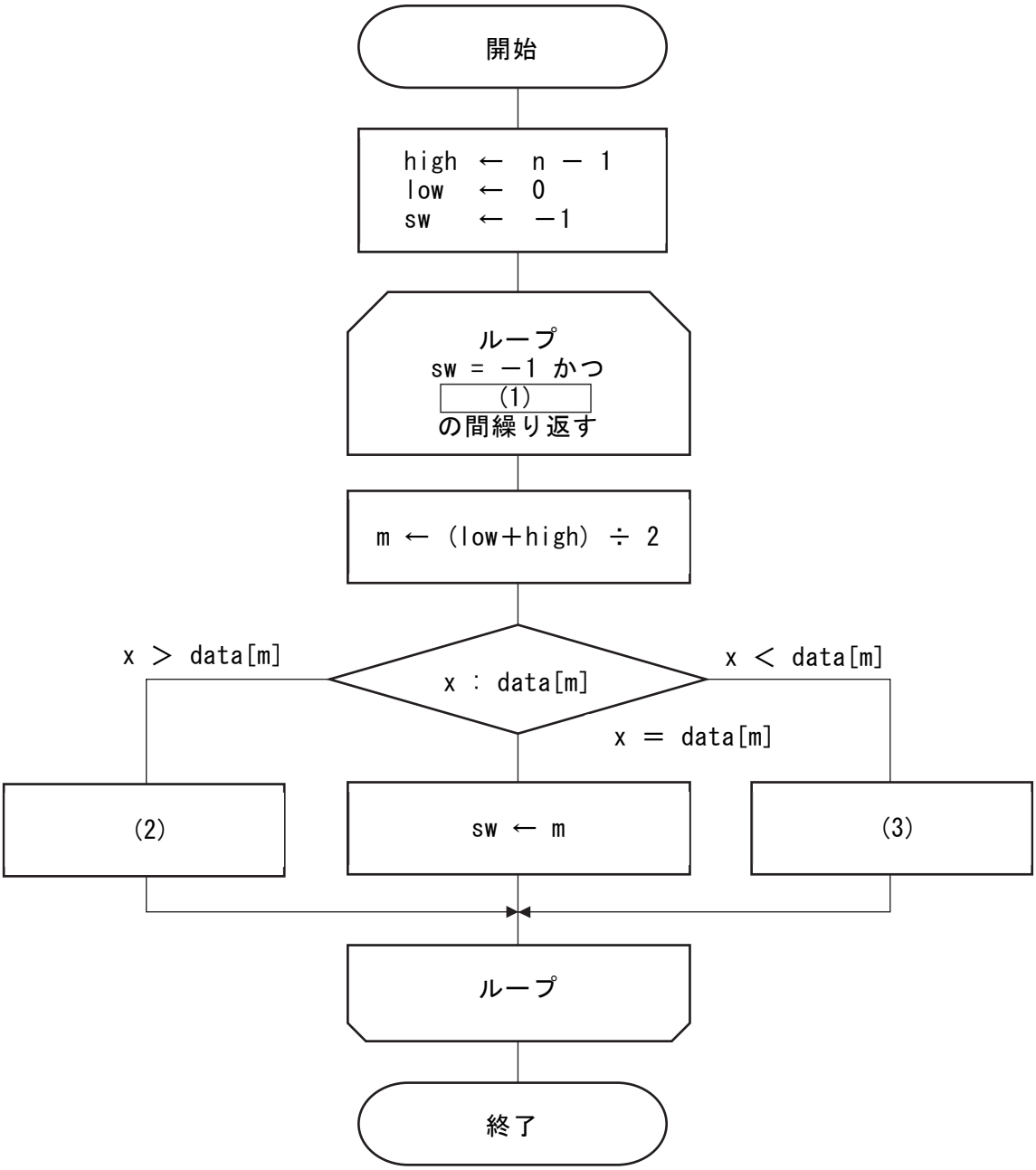

図３　二分探索法によるプログラムの流れ図

［プログラム］

○整数型: binarySearch(整数型の配列:data, 整数型:n, 整数型:x)

```
整数型: high, low, sw
high ← n - 1
low  ← 0
sw   ← -1

while( sw が -1 と等しい and [ (1) ] )

    整数型:m
    m ← (low + high) ÷ 2

    if ( [ (4) ] )
        sw ← m
    elseif ( [ (5) ] )
        [ (2) ]
    else
        [ (3) ]
    endif

endwhile

return sw
```

(1) の解答群

ア. low が high より小さい　　イ. low が high より大きい

ウ. low が high 以下　　エ. low が high 以上

(2), (3)の解答群

ア. high ← m - 1　　イ. high ← m + 1

ウ. low ← m - 1　　エ. low ← m + 1

(4), (5)の解答群

ア. data[m] が x でない　　イ. data[m] が x と等しい

ウ. data[m] が x より小さい　　エ. data[m] が x より大きい

＜設問２＞　次の二分探索法に関する記述を読み，記述中の［　　］に入れるべき適切な字句を解答群から選べ。

二分探索法では一度の比較で次の探索範囲がほぼ半分になる。このことから，一般的にデータ数がｎの場合，最大比較回数は，「$\log_2 n$ を超える最小の整数」となる。例えばデータ数が 10 の場合，$\log_2 10 < \log_2 16 = \log_2 2^4 = 4\log_2 2 = 4 \times 1 = 4$（回）である（$\log_2 2$ は１）。同様にデータ数が 60 の場合は $\log_2 60 < \log_2$［　(6)　］から最大比較回数は６回となる。

また，最大比較回数がｐ回であった配列がある場合，この配列の大きさを２倍にした場合の最大比較回数は［　(7)　］回になる。

(6) の解答群

ア．64　　イ．128　　ウ．256　　エ．512

(7) の解答群

ア．p　　イ．p ＋ 1　　ウ．2p － 1　　エ．2p

＜メモ欄＞

＜メモ欄＞

＜メモ欄＞