文部科学省後援

情報システム試験

令和８年度前期 情報検定

＜実施　令和8年9月13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21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，デジタル「合否通知」，デジタル「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。

単方向リストは，データを格納する要素と，次の要素の場所を示すポインタをセットで管理するデータ構造である。データの追加や削除をポインタのつなぎ替えだけで行えるため，データ処理を効率的に行うことができる。

[単方向リストの説明]

本問では，単方向リスト list を配列を用いた構造体で管理し，次のように表記する。なお，添字は 0 から始まる。

・list[i].data：配列の i 番目の要素のデータ

・list[i].next：配列の i 番目の要素の次ポインタ（次の要素の添字）

先頭の要素の添字は変数 head に格納されているため，list[head].data と記述することで先頭のデータにアクセスできる。

なお，単方向リスト list には十分な領域があり，１つ以上のデータが格納されているものとする。単方向リスト list の終端データの次ポインタには-1 が格納されている。

[変数の説明]

head：リストの先頭要素が格納されている配列 list の添字。

del：単方向リストから探し出して削除したいデータを格納する変数。事前に入力されているものとする。

i：リストを先頭から順にたどるための，現在調べている要素の添字を格納する変数。

prev：要素を削除する際の次ポインタのつなぎ替えに使用する変数。現在調べている要素 list[i]の１つ前の要素の添字を保持する。

＜設問１＞　次のデータ処理に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

単方向リストにおけるデータの挿入と削除の仕組みについて考える。初期状態として，表のようにデータが格納されている。なお，head には 0 が格納されている。

表　単方向リスト（初期状態）

添字 i	0	1	2	3	…
list[i].data	10	30	20		…
list[i].next	2	-1	1		…

データの挿入では，データ「10」と「20」の間に，新しいデータ「15」を挿入する。新しいデータは空いている添字 3 の要素に格納する。挿入処理では，まず list[3].data に 15 を格納し，次にポインタをつなぎ替える。具体的には，元の順番である「10 → 20」を「10 → 15 → 20」にするため，list[3].next には［ (1) ］が格納され，list[0].next には［ (2) ］が格納される。

次にデータの削除では，上記で挿入した後の状態から，データ「20」を削除する。データ「20」は添字 2 の要素に格納されている。これをリストから切り離すためには，その直前の要素であるデータ「15」の次ポインタをつなぎ替える必要がある。処理の結果，list[3].next には［ (3) ］が格納される。

(1) ～ (3) の解答群

ア．-1　　イ．0　　ウ．1

エ．2　　オ．3

＜設問２＞　次の流れ図の［　　］に入れるべき適切な字句を解答群から選べ。

図は単方向リストから削除したいデータがあった場合に削除する処理を表した流れ図である。

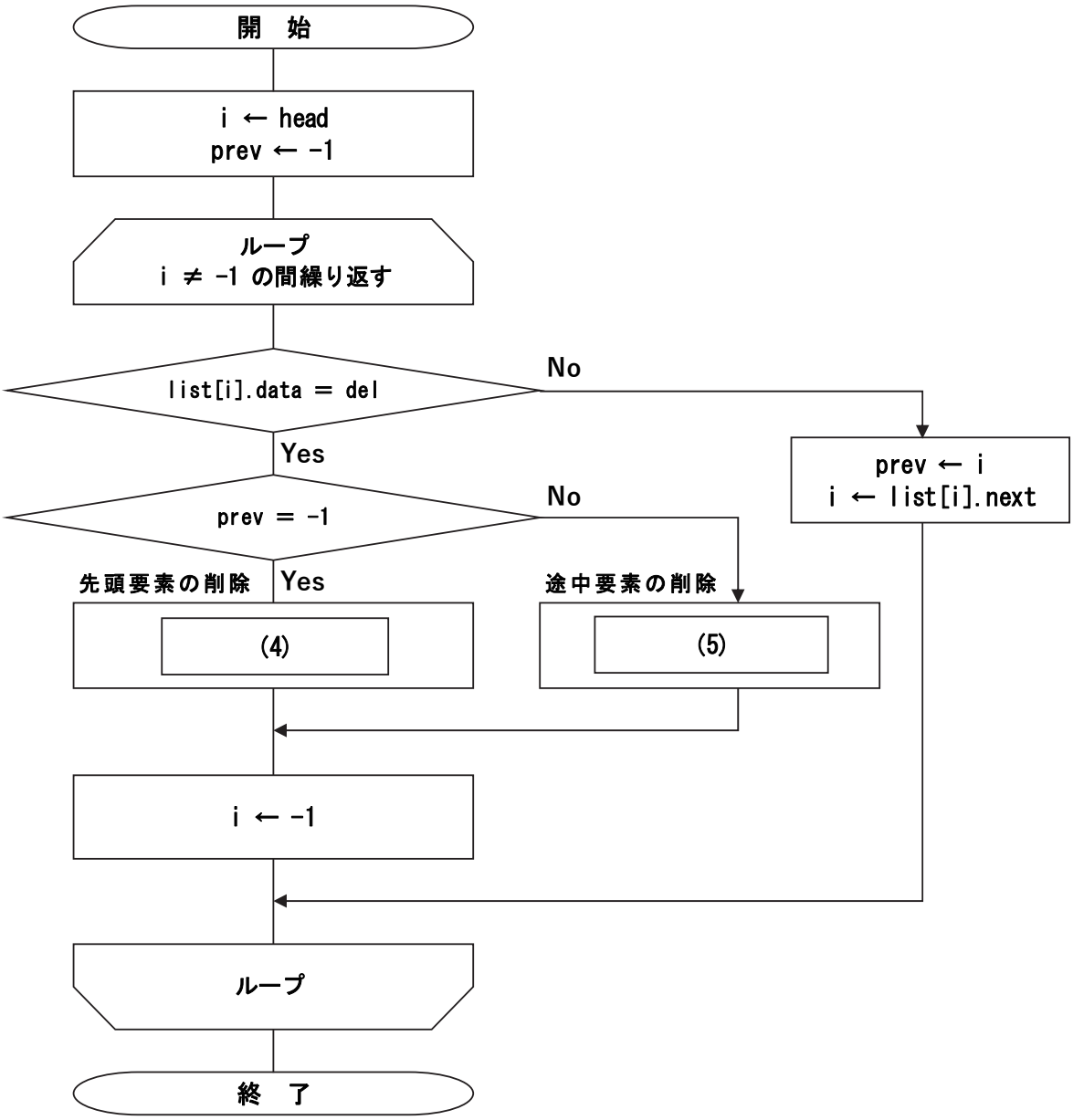


図　流れ図

(4) の解答群

ア. head ← list[i].data　　イ. head ← list[i].next

ウ. prev ← -1　　エ. prev ← list[i].next

(5) の解答群

ア. list[i].data ← -1

イ. list[i].next ← list[prev].next

ウ. list[prev].data ← list[i].data

エ. list[prev].next ← list[i].next

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。

入力された任意の文字列について，どの文字が何回使われているかの頻度（出現回数）をカウントし，集計結果を配列に記録する処理を行う。

［流れ図の説明］

文字列を先頭から１文字ずつ取り出して，それが最初に現れた文字であれば新しく登録し，初期値として１を設定する。すでに登録されている文字であれば，その文字が現れる毎に出現回数をカウントしていく。文字列内のすべての文字についてこの処理を繰り返すことで，使用されているすべての文字の種類と出現回数を求めることができる。なお，添字は０から始まるものとする。

［変数の説明］

・str：入力された文字列を格納する配列。（１文字目は str[0]と表記する）
・len：入力された文字列の文字数。
・charlist：文字列から見つけ出した文字を登録する配列。
・countlist：登録された各文字の出現回数を記録する配列。たとえば，charlist[j]に登録された文字の出現回数は，countlist[j]に格納される。
・types：これまでに新しく見つかり，登録された文字数。
・i：文字列 str を先頭から何文字目まで調べたかを表す添字。
・j：charlist に登録済みの文字を検索するための添字。
・c：文字列 str から現在取り出して調べている１文字を格納する変数。
・found：取り出した文字 c が，すでに登録済みかどうかを判定するための論理型変数。見つかれば true，見つからなければ false を格納する。

［流れ図］

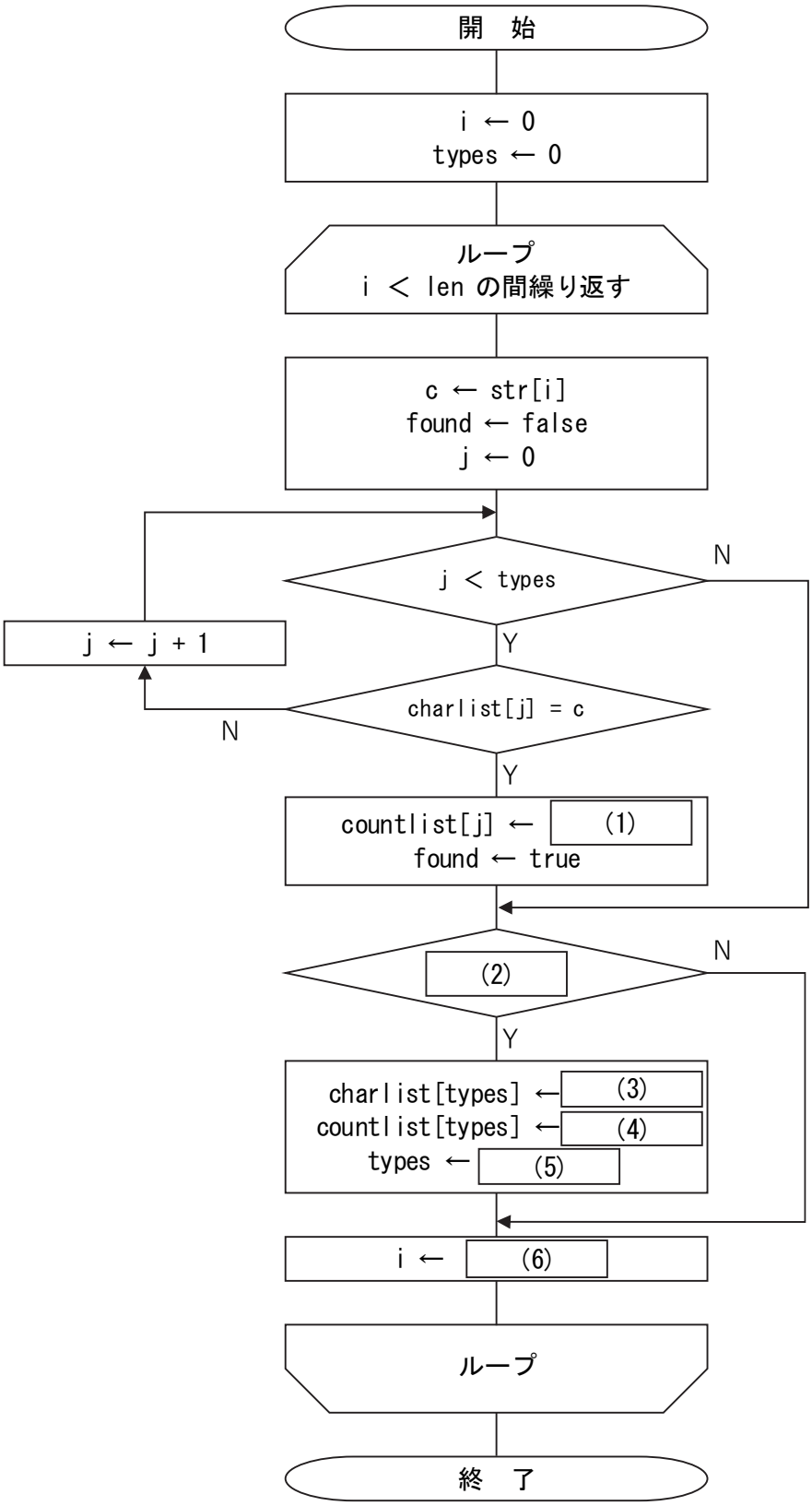

開　始
i ← 0
types ← 0
ループ
i ＜ len の間繰り返す
c ← str[i]
found ← false
j ← 0
j ＜ types
N
Y
j ← j + 1
charlist[j] = c
N
Y
countlist[j] ← (1)
found ← true
(2)
N
Y
charlist[types] ← (3)
countlist[types] ← (4)
types ← (5)
i ← (6)
ループ
終　了

図　流れ図

(1) の解答群

ア. countlist[i] + 1　　イ. countlist[j]

ウ. countlist[j] + 1　　エ. countlist[types] + 1

(2) の解答群

ア. found = false　　イ. found = true

ウ. i ＜ len　　エ. j ＜ types

(3) の解答群

ア. c　　イ. i

ウ. j　　エ. str[j]

(4) の解答群

ア. 0　　イ. 1

ウ. c　　エ. i

(5) の解答群

ア. i + 1　　イ. j + 1

ウ. types　　エ. types + 1

(6) の解答群

ア. i　　イ. i + 1

ウ. j + 1　　エ. types + 1

問題３　次の並べ替えに関する記述を読み，各設問に答えよ。

複数のデータをある規則に従って並べ替えるアルゴリズムには，選択法ソート，バブルソート，クイックソートなどがある。ここでは，配列 ary に格納されたデータを昇順に並べ替えることを考える。なお，配列の添字は１から始まり，要素数は変数ｎに与えられる。

［選択法ソートについて］

未整列部分から最小値を探し，それを未整列部分の先頭と交換する操作を繰り返して整列を行う。処理概要は次のようになる。

１．未整列部分の中から最小値を探す。

２．未整列部分の先頭位置にある要素と，最小値が格納されている位置の要素を交換する。

３．整列済みの要素を除いた残りの部分を新たな未整列部分として，上記１と２の作業を未整列部分の要素数が１になるまで繰り返す。

＜設問１＞　次の配列 ary を選択法ソートで昇順に整列する流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

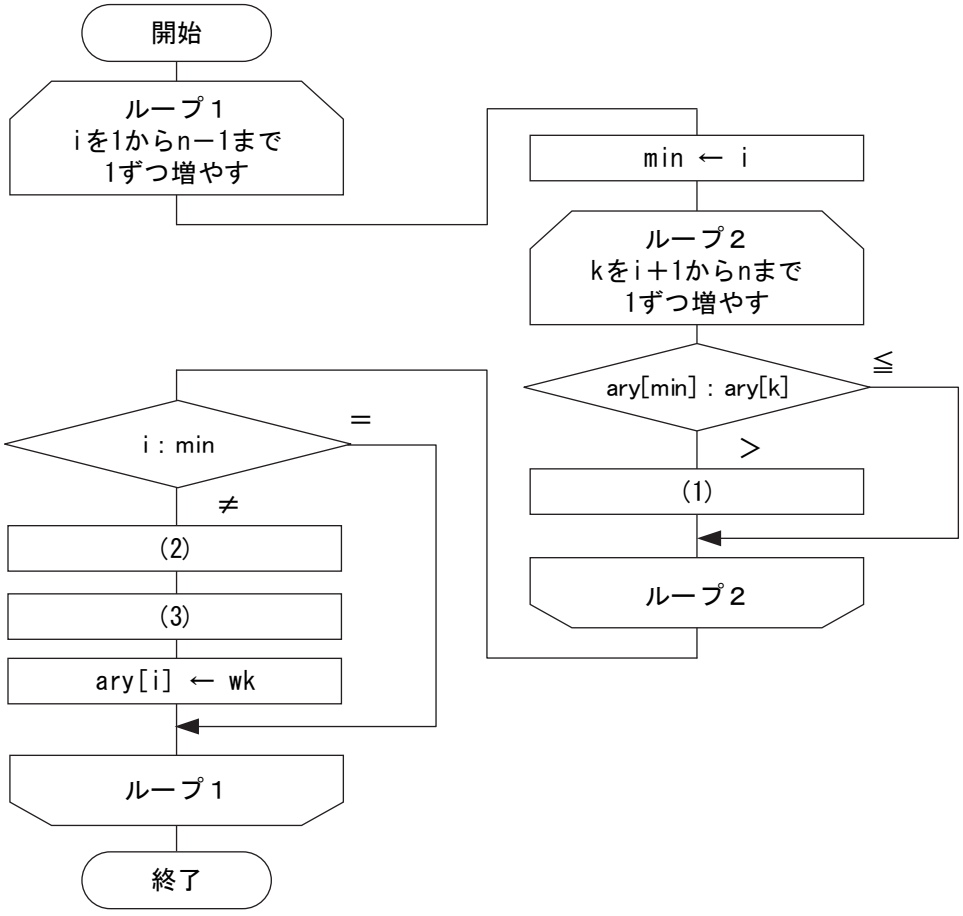


図１　選択法ソート

(1) ～ (3) の解答群

ア．min ← i　　イ．min ← k
ウ．ary[min] ← ary[i]　　エ．ary[min] ← ary[k]
オ．wk ← ary[min]　　カ．ary[i] ← ary[min]
キ．ary[k] ← ary[min]　　ク．ary[min] ← wk

＜設問２＞　次の図１の流れ図の動作に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。なお，図１の流れ図の(1)～(3)は正しい処理が入っているものとする。

配列 ary の要素および要素数 n が図２の内容で図１の流れ図を実行するとき，図１の流れ図中のループ２の処理を最初に終えたときの min の値は［(4)］である。

	1	2	3	4	5		n
ary	15	13	11	12	14		5

図２　開始時の値

(4) の解答群

ア．1　　イ．2　　ウ．3　　エ．4　　オ．5

＜設問３＞　次の選択法ソートの実行回数に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。なお，図１の流れ図の(1)～(3)は正しい処理が入っているものとする。

図１の流れ図において，ループ２の最初に処理する比較「ary[min]：ary[k]」の実行回数は，選択法ソートでは常に一定である。要素数を n としたとき，図１の流れ図の場合，［(5)］回の比較が行われる。

(5) の解答群

ア．n　　イ．n(n － 1) ÷ 2
ウ．n(n ＋ 1) ÷ 2　　エ．n^2

＜設問４＞　次のバブルソートに関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

［バブルソートについて］

バブルソートは，配列の中で隣り合う要素を比較し，並べ替えの規則に合わない場合は要素を交換するという操作を繰り返すことで整列を行うものである。昇順に並べ替える場合は，比較する要素の位置の小さい方が小さい値となる。

ここで，昇順に整列する場合を考える。未整列部分の先頭から末尾まで，隣り合う要素の比較と交換を行う一連の操作を１回行うと，未整列部分の中で［(6)］が配列の末尾に移動する。例えば，配列の要素数を５とする時，この一連の操作を最大でも［(7)］回繰り返すことで並べ替えが完了する。なお，並べ替えが確定する要素は［(8)］順である。

(6) の解答群

ア．最小値　　イ．最大値
ウ．最後に比較した値　　エ．先頭の値

(7) の解答群

ア．3　　イ．4　　ウ．5　　エ．6

(8) の解答群

ア．値の大きい　　イ．値の小さい
ウ．比較回数の多い　　エ．要素の位置が小さい

＜設問５＞　次の配列 ary をバブルソートにより昇順に整列する流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

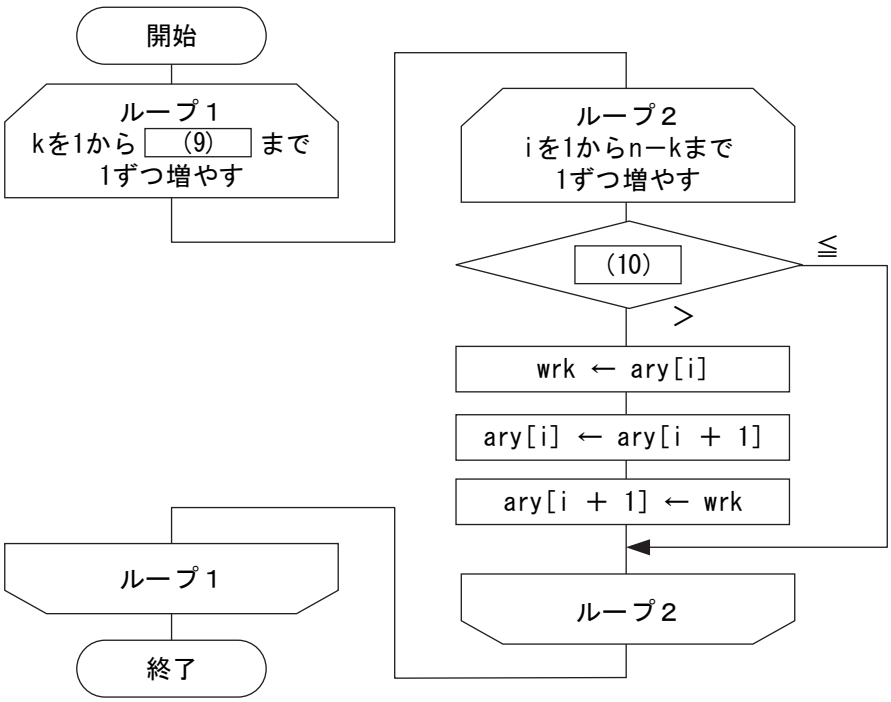


図３　バブルソートの流れ図

(9) の解答群

ア．n － 1　　イ．n

ウ．n ＋ 1　　エ．n^2

(10) の解答群

ア．ary[i] ： ary[i － 1]　　イ．ary[i － 1] ： ary[i]

ウ．ary[i] ： ary[i ＋ 1]　　エ．ary[i ＋ 1] ： ary[i]

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1から始まる。

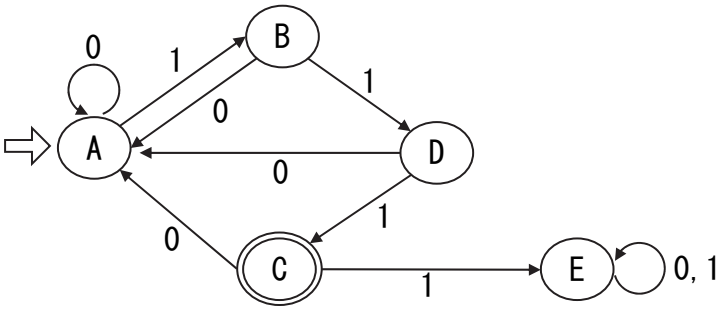
手続 StateTD は，図の状態遷移図に入力値(8 ビット)が与えられた場合，初期状態である節 A から遷移順に節記号を出力し，最終的に節 C で終わる場合は戻り値を true に，そうでない場合は戻り値を false にする。

表　図の状態遷移図に与えられる入力値と節の遷移順及び戻り値の値

入力値	節の遷移順	戻り値
{1,1,0,0,0,1,1,1}	A,B,D,A,A,A,B,D,C	true
{0,0,0,1,1,0,1,1}	A,A,A,A,B,D,A,B,D	false
{1,0,1,1,1,1,0,0}	A,B,A,B,D,C,E,E,E	false

下図の状態遷移図を配列で表現したのが Node と Trans である。例えば，節 C の状態から遷移するのは，入力値が 0 の時，Trans[3][1]が示す節 A に遷移し，入力値が 1 の時は Trans[3][2]が示す節 E に遷移する。

また，入力値は配列 Bin に 1 ビットずつ，0 または 1 の値が格納される。



		1
Node	1	A
	2	B
	3	C
	4	D
	5	E

		1	2
Trans	1	1	2
	2	1	4
	3	1	5
	4	1	3
	5	5	5

図　状態遷移図と配列 Node，配列 Trans の内容

＜設問１＞　手続 StateTD を呼び出すプログラム中の［　　］に入れるべき適切な字句を解答群から選べ。

［プログラム］

```
大域：文字型の配列：Node ← {"A", "B", "C", "D", "E"}
大域：整数型の配列：Trans ← {{1, 2}, {1, 4}, {1, 5}, {1, 3}, {5, 5}}

○論理型：StateTD (整数型の配列：Bin)
  整数型：i, k
  論理型：result ← false

  k ← 1
  Node[k]を出力
  for (i を 1 から 8 まで 1 ずつ増やす)
      if (Bin[i] が 0 と等しい)
          (1)
      else
          (2)
      endif
      Node[k]を出力
  endfor

  if ( (3) が "C" と等しい)
      result ← true
  endif
  return result
```

(1)，(2) の解答群

ア．k ← Trans[i][1]　　イ．k ← Trans[i][2]

ウ．k ← Trans[k][1]　　エ．k ← Trans[k][2]

(3) の解答群

ア．Node[i]　　イ．Node[k]

ウ．Trans[i][k]　　エ．Trans[k][i]

＜設問２＞　次のプログラムの実行に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

入力値 Bin に{0,1,0,1,0,1,1,1}が入力され手続 StateTD を実行した場合，節の遷移順として［　(4)　］が出力され，result の値には［　(5)　］が設定される。

(4) の解答群

ア．A,A,B,A,B,A,B,D,C

イ．A,A,B,D,C,E,E,E,E

ウ．A,B,D,A,B,D,A,A,B

(5) の解答群

ア．**false**　　イ．**true**

問題５　次のヒープに関する記述を読み，各設問に答えよ。なお，本問題における配列の添字は 0 から始める。

［木構造について］

木構造とは，データの並びを，根（ルート），枝（エッジ），節（ノード）といった木（ツリー）の要素に例えて，階層的に管理するデータ構造である。最上位に位置する節を根と呼び，ある節から直接つながって下位に位置する節を子，上位に位置する節を親と呼ぶ。さらに，子を持たない節を葉（リーフ）と呼ぶ。根から各節までの段数をその節の高さといい，根から最も深い葉までの段数を木全体の高さという。

各節が持つ子の数が０～２個である木を二分木という。さらに，最下段を除くすべての段が完全に埋まっており，最下段の節が左から順に配置されている二分木を完全二分木という。完全二分木の例を図１に示す。

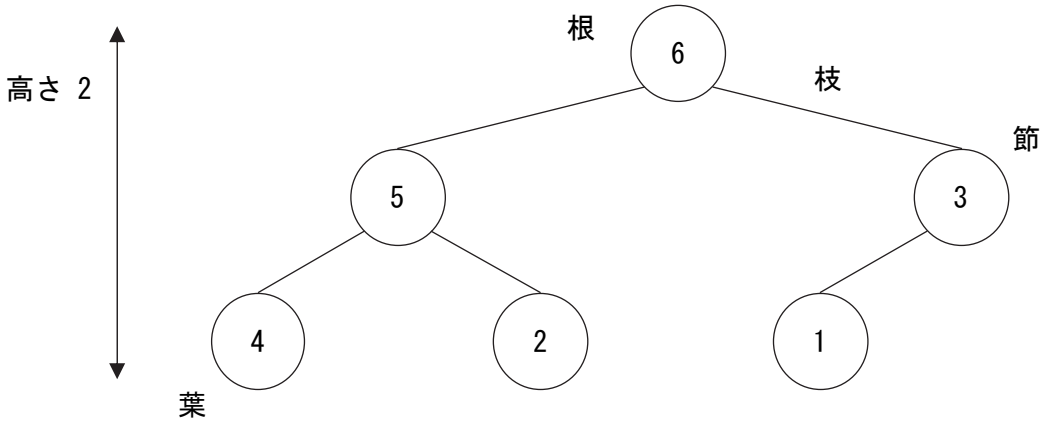


図１　完全二分木の例

図１のように，完全二分木は，根から順に左から右へ規則的に節が配置される構造である。この性質により，配列のデータ構造による表現が可能である。例えば，図１の完全二分木を配列 t で表現すると配列 t［6，5，3，4，2，1］となる。このとき，配列の添字を i とすると，左の子は 2i + 1，右の子は 2i + 2 の位置に対応する。

［ヒープについて］

完全二分木のうち，親の節の値が常に子の節の値以上（または以下）という条件を満たすデータ構造をヒープという。木の親子関係に順序条件を持つことが特徴であり，親の値が子の値以上であるものを最大ヒープ，親の値が子の値以下であるものを最小ヒープという。

本問題では，最大ヒープを扱う。図１の完全二分木は，配列 t［1，2，3，4，5，6］をヒープ構造で表したものである。

＜設問１＞　次のヒープの構築に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

配列 r［2, 4, 5, 1, 3, 6］からヒープを構築することを考える。ヒープでは以下２つの手順を繰り返す。

手順①　処理対象となる親を決める

配列 r におけるすべての親を対象として，最後の親（配列の添字が最も大きい親）から最初の親（配列の添字が最も小さい親）へと順に処理を行う。なお，最初に選択する親の添字は「(配列の要素数 ÷ 2) － 1」で計算できる（結果は小数点以下切り捨て)。配列 r の場合，要素および節の値が［ (1) ］のものを親として比較を始める。

手順②　比較対象の親とその子要素の交換を行う

手順①で選ばれた親とその子要素を比較し，最大値が親の位置に格納されるようにデータを交換する。子要素の方が親よりも大きく，交換が発生した場合は，交換後の子の位置を新たな親の位置として，同様の操作を続ける。交換が発生しなければ，その親に対する処理を終了する。

例えば，配列 r の場合，［ (1) ］および，その子要素の中の最大値［ (2) ］を比較する。子要素の方が親よりも大きいため［ (1) ］と［ (2) ］を交換し，子の位置を新たな親の位置として操作を続ける。しかし，新たな親の位置から見た子の位置は配列の範囲を超えているため，これ以上の交換は発生しない。このときの配列 r は［ (3) ］である。

以上の手順を残りの親についても同様に繰り返す。ヒープを構築した後の配列 r は［ (4) ］となる。

(1)，(2) の解答群

ア．3　　イ．4　　ウ．5　　エ．6

(3)，(4) の解答群

ア．[2, 4, 6, 1, 3, 5]　　イ．[2, 5, 6, 3, 4, 1]

ウ．[6, 2, 4, 5, 1, 3]　　エ．[6, 4, 5, 1, 3, 2]

＜設問２＞　次のヒープソートに関するプログラム中の［　　］に入れるべき適切な字句を解答群から選べ。

［ヒープソートの説明］

ヒープが構築された配列では，配列の先頭には常に最大値が格納される。この性質を利用すると，最大値を順に確定することにより配列を整列できる。これをヒープソートと呼ぶ。この方法は，最大値を順に確定する点で基本選択法と同じ考え方であるが，最大値の探索をヒープ構造によって効率的に行う点が特徴である。

昇順に整列する場合，最大値と最後尾のデータを入れ替えて最大値を確定する。この操作によってヒープの親子関係が部分的に崩れるため，残りの要素に対してヒープを再構築する。図２のように「最大値の確定 → ヒープ再構築 → 最大値の確定 → ヒープ再構築 → …」という操作を，配列の要素がすべて確定するまで繰り返す。

①［ヒープ］　　ヒープ構築後の配列を用意する，　h[0]　は最大値である

添字	0	1	2	3	4	5
h	6	5	3	4	2	1

②［最大値の確定］　最大値　h[0]　と　最後の値　h[5]　を交換する

添字	0	1	2	3	4	5
h	1	5	3	4	2	6

③［ヒープ再構築］　h[0] ～ h[4]　でヒープを再構築する

添字	0	1	2	3	4	5
h	5	4	3	1	2	6

④［最大値の確定］　最大値　h[0]　と　最後の値　h[4]　を交換する

添字	0	1	2	3	4	5
h	2	4	3	1	5	6

⑤［ヒープ再構築］　h[0] ～ h[3]　でヒープを再構築する

添字	0	1	2	3	4	5
h	4	2	3	1	5	6

⑥　以降，　最大値の確定　と　ヒープ再構築　の手順を繰り返す

図２　ヒープソートの例

［プログラムの説明］

ランダムな値が格納された大域変数の配列 h をヒープソートにより並べ替える手続 heapSort である。この手続は，最初にヒープの構築を行い，その後，最大値の確定とヒープ再構築の手順を繰り返しながら配列 h の内容を昇順に整列する。手続 heapSort では図３に示す３つの手続呼び出しを行いヒープソートする。

［ 手続makeHeapの仕様 ］

要素数 heapSize の 配列 h をヒープ構築する。

引数 / 返却値	データ型	意味
引数	整数型 heapSize	配列 h の操作対象となる要素数
返却値		なし

［ 手続downHeapの仕様 ］

配列 h のヒープを再構築する。

引数 / 返却値	データ型	意味
引数	整数型 heapSize	配列 h の操作対象となる要素数
	整数型 p	配列 h の操作対象となる親の添字
返却値		なし

［ 手続swapの仕様 ］

要素を交換する。

引数 / 返却値	データ型	意味
引数	整数型 i	交換する値
	整数型 j	交換する値
返却値		なし

図３　各手続の仕様

［プログラム］

```
大域: 文字型の配列: h ← {2, 4, 5, 1, 3, 6}

○heapSort()
  整数型:heapSize
  heapSize ← 配列 h の要素数
  makeHeap( heapSize )
  while( [ (5) ] )
      swap( 0, heapSize − 1 )
      heapSize ← heapSize − 1
      downHeap( heapSize, 0 )
  endWhile

○makeHeap( 整数型: heapSize )
  【 要素数 heapSize の 配列 h をヒープを構築する。 】

○downHeap( 整数型: heapSize, 整数型: p )
  【 配列 h のヒープを再構築する。 】

○swap( 整数型: i, 整数型: j )
  整数型:temp
  temp    ← h[i]
  [ (6) ]
  h[j]    ← temp
```

(5) の解答群

ア. heapSize が 1 より大きい　　イ. heapSize が 2 より大きい

ウ. heapSize が 1 より小さい　　エ. heapSize が 1 と等しい

(6) の解答群

ア. h[i] ← h[j]　　イ. h[i] ← h[j + 1]

ウ. h[j] ← h[i]　　エ. h[j] ← h[i + 1]