文部科学省後援

情報システム試験

令和６年度後期 情報検定

＜実施　令和7年2月9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19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。筆記用具が破損等により使用不能となった場合は，手をあげて試験監督者に合図してください。

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，設問に答えよ。

データ構造の１つに単方向リストがある。単方向リストはデータを要素とポインタのセットで管理することで，データの追加や削除を高速に行うことができる。

［流れ図の説明］

単方向リスト List はデータと次ポインタをセットにした構造体で管理し，次のように表記する。

List の０番目のデータにアクセスする場合：List[0].data

List の０番目の次ポインタにアクセスする場合：List[0].next

先頭を指し示すポインタは変数 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することができる。手順としては，目的のデータと List の data が一致するまで next のポインタに移動していくことで目的のデータにたどり着くことができる。削除処理は，１つ前のデータを参照する変数 prev を利用して実現している。

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。

＜設問＞　次の削除処理の流れ図の［　　］に入れるべき適切な字句を解答群から選べ。

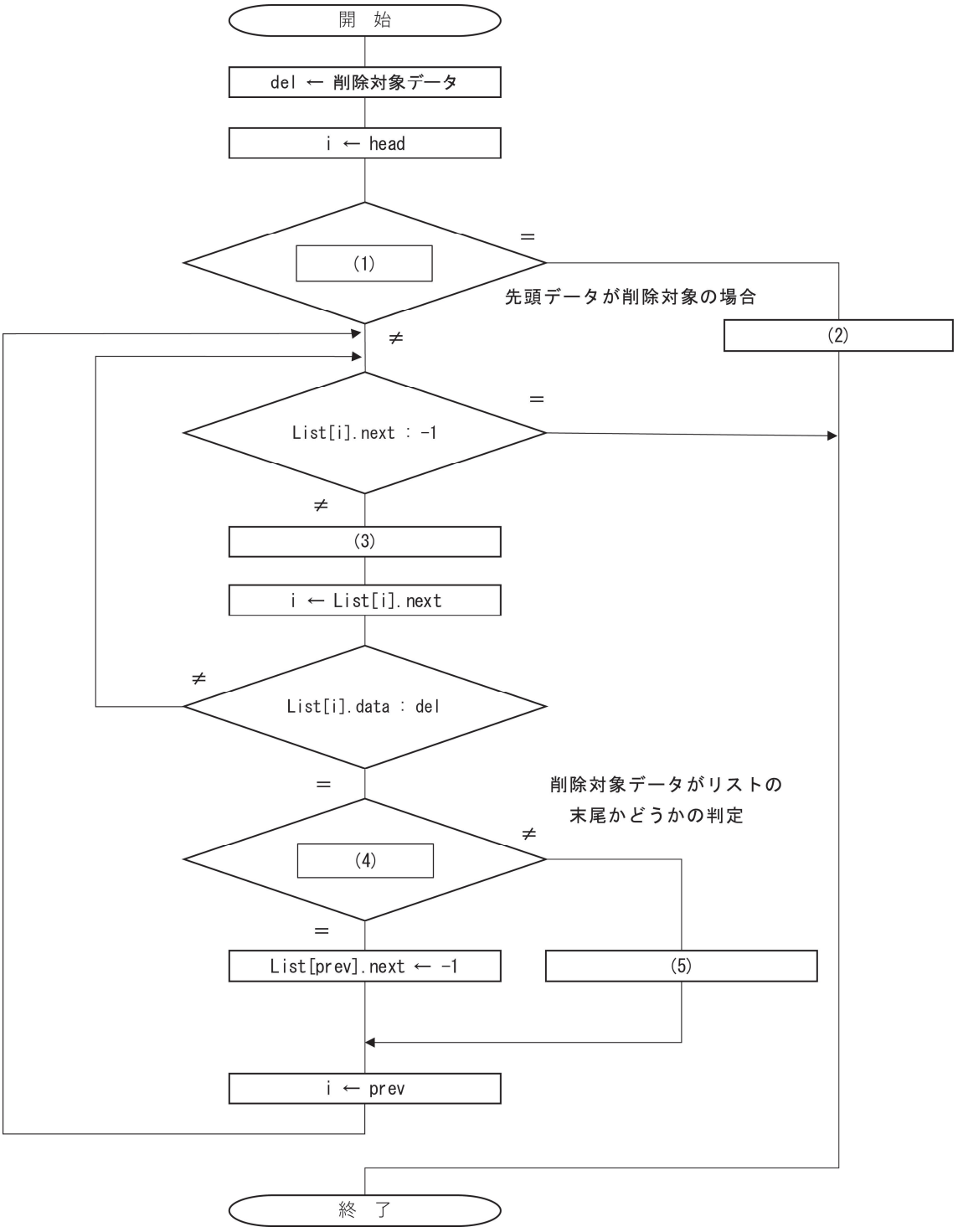

図　流れ図

(1) の解答群

ア. List[i].data : -1　　イ. List[i].data : del
ウ. List[i].data : head　　エ. List[i].data : i

(2) の解答群

ア. head ← List[i].data　　イ. head ← List[i].next
ウ. prev ← -1　　エ. prev ← del

(3) の解答群

ア. i ← del　　イ. i ← prev
ウ. prev ← del　　エ. prev ← i

(4) の解答群

ア. List[i].data : -1　　イ. List[i].data : del
ウ. List[i].next : -1　　エ. List[i].next : del

(5) の解答群

ア. List[i].data ← -1
イ. List[i].prev ← List[prev].next
ウ. List[prev].next ← List[i].next

問題２　次の二分探索法に関する記述を読み，各設問に答えよ。

［二分探索法の説明］

二分探索法は，整列済みの一次元配列からデータを探索する方法である。なお，配列の大きさは n に，探したいデータは x に，探索する一次元配列は，dat[0]～dat[n－1]に昇順に格納済みとする。流れ図で使用する変数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 とする。なお，初期値は，low＝0，high＝n－1 である。

②　探索範囲の中央要素となる dat[m]と比較する。ただし，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 dat[m]＝x なら，見つかったので指定の処理を実行後終了する。

Ⅱ dat[m]＜x なら，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。

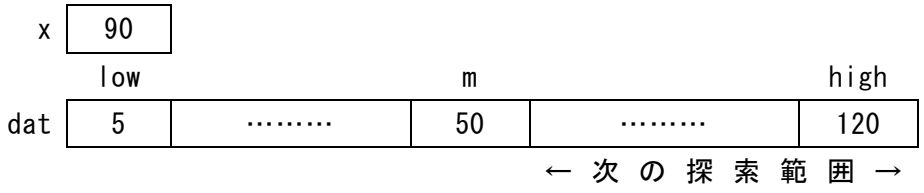

図１　比較例１

Ⅲ dat[m]＞x なら，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。

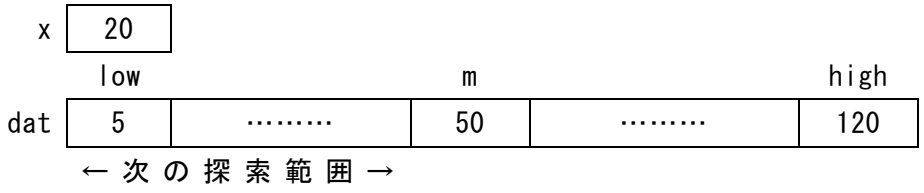

図２　比較例２

③　low＞high または dat[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 dat に存在しないことになる。

＜設問１＞　二分探索法の説明を読み，図３の流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。なお，流れ図中の除算は小数点以下を切り捨てる。

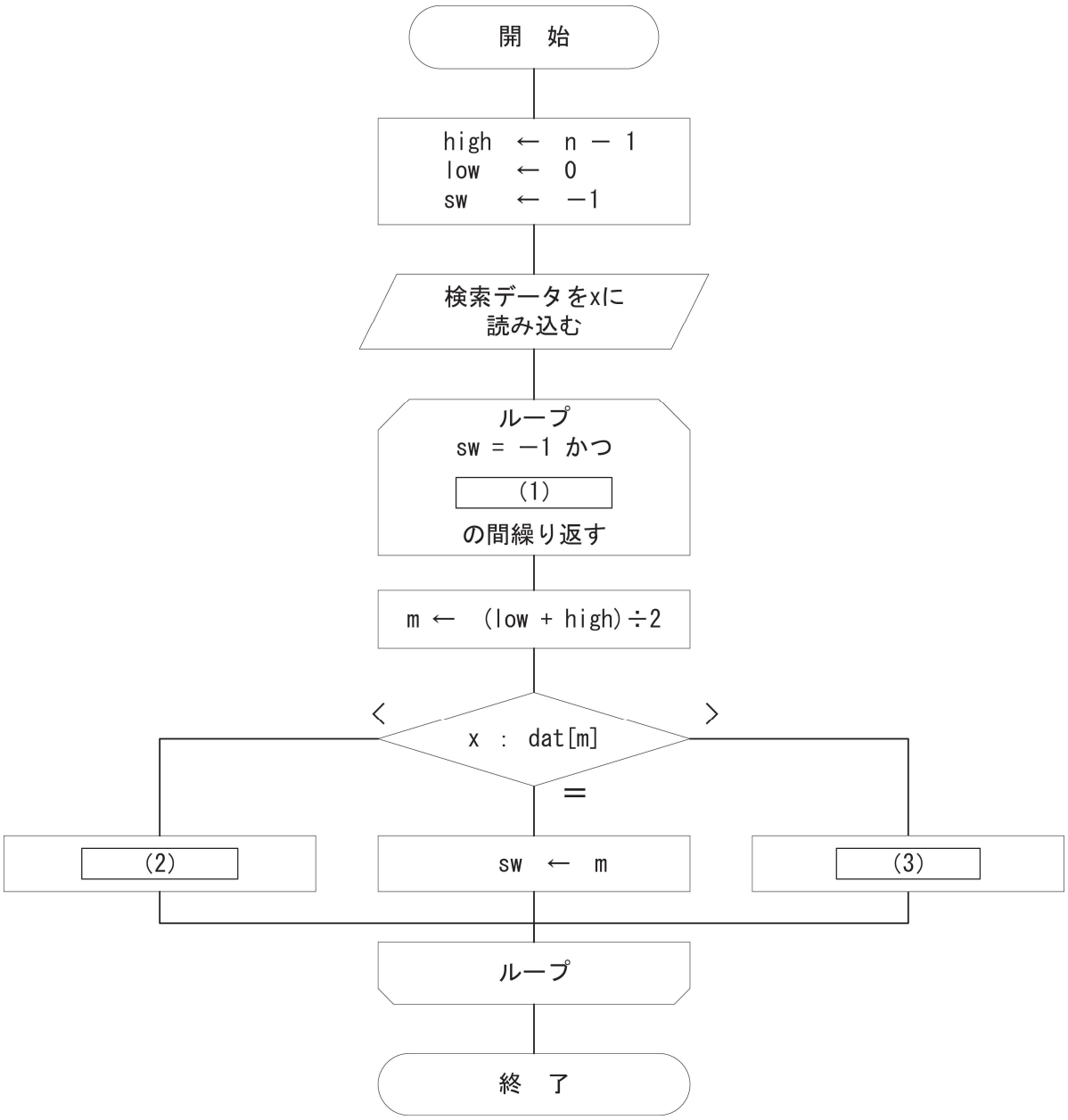

図３　二分探索法の流れ図

(1) の解答群

ア．low ＜ high　　イ．low ≦ high

ウ．low ＞ high　　エ．low ≧ high

(2), (3)の解答群

ア．high ← m − 1　　イ．high ← m ＋ 1

ウ．low ← m − 1　　エ．low ← m ＋ 1

＜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」と言われている。Log の値を整数で考えると，$\log_2 2=1$，$\log_2 4=2$，$\log_2 8=3$，$\log_2 16=4$ である。$\log_2 10$ は，$\log_2 8$ と $\log_2 16$ の間の値であるから３と４の間の値であることがわかる。したがってデータ数が 10 の場合，「$\log_2 10$ を超える最小の整数」は４である。同様にデータ数が 100 の場合は $\log_2 100 <$ $\log_2$［　(4)　］から最大比較回数は７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［　(5)　］回になる。

(4) の解答群

ア．64　　イ．128　　ウ．256　　エ．512

(5) の解答群

ア．p　　イ．p ＋ 1　　ウ．2p － 1　　エ．2p

問題３　次の文字列処理に関する記述を読み，各設問に答えよ。

［文字列処理について］

配列の先頭から順番に１文字ずつ格納したデータを文字列として扱う。なお，配列の添字は１から始まるものとし，配列の大きさは実行に必要な大きさが確保されているものとする。また，記述中で使われている除算は小数点以下を切り捨てる。

＜設問１＞　次の文字列の反転に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

文字列の中央を中心にして左右に反転させるには，配列の先頭から末尾に向けて順番に参照する添字と，配列の末尾から先頭に向けて順番に参照する添字を使って交換する方法がある。ここで，次のように変数を設定する。

・t　　：文字列を格納した配列
・tLen　：配列に格納した文字数
・left　：配列の先頭から末尾に向けて順番に要素を参照するための添字
・right：配列の末尾から先頭に向けて順番に要素を参照するための添字

この時，left の初期値は［　(1)　］であり，right の初期値は［　(2)　］である。初期設定後に次の３つの処理を繰り返すことで文字列を反転する。

① t[left]と t[right]の要素を交換する。
② left に１を加える。
③ right から１を減じる。

これらの処理は，［　(3)　］回繰り返す。

(1) ～ (3) の解答群

ア．0	イ．1	ウ．left	エ．tLen － 1
オ．tLen	カ．tLen ＋ 1	キ．tLen ÷ 2	ク．right

＜設問２＞　次の設問１とは異なる方法に関する記述中の［　　］に入れるべき適切な字句を解答群から選べ。

添字の初期値を配列のほぼ中央に設定し，添字の値を先頭方向および末尾方向に向かって変化させながら処理する場合を考える。ここで，文字列を配列 t に，その文字数を tLen に格納している状態で，文字数が４および５の場合について考える。

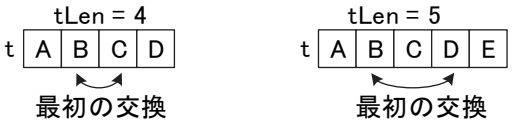

図１　文字数が４の場合と５の場合の交換

ほぼ中央の位置から交換を始めるので，文字数が４の場合は［　(4)　］，文字数が５の場合は［　(5)　］を最初に交換する。ここで，文字数が偶数か奇数かによって，最初に交換する相手が異なってくることに注目する。

ほぼ中央の位置から要素位置の小さい方へ順番に参照する添字を left，要素位置の大きい方へ順番に参照する添字を right とした時，それぞれの初期値は次の計算により求める。なお，「a　mod　b」は a を b で割った剰余を求めるものである。

・left　：tLen ÷ 2
・right：［　(6)　］

(4)，(5) の解答群

ア．t[0] と t[2]　　イ．t[0] と t[3]　　ウ．t[1] と t[3]
エ．t[1] と t[4]　　オ．t[2] と t[3]　　カ．t[2] と t[4]

(6) の解答群

ア．left ＋ 1　　イ．left ＋ (tLen mod 2)
ウ．left ＋ (tLen mod 2) － 1　　エ．left ＋ (tLen mod 2) ＋ 1

＜設問３＞　次の流れ図は設問２の考え方で文字列を反転するものである。流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。なお，文字列は配列 t に 1 文字以上の文字列が配列の先頭から順番に格納されており，文字数は tLen に格納されている。また，空欄(6)は設問２と同じ式が入る。

［流れ図］

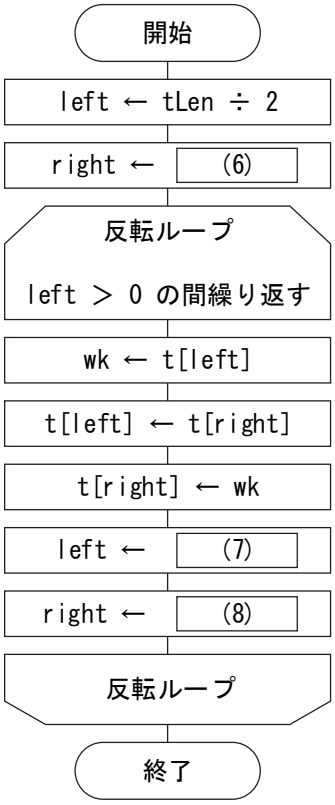

図２　文字列を反転する流れ図

(7)，(8) の解答群

ア．left ＋ 1　　イ．left － 1

ウ．right ＋ 1　　エ．right － 1

オ．(left ＋ right) ÷ 2　　カ．left ＋ right － 1

＜設問４＞　次の文字列の挿入処理に関する記述を読み，流れ図中の［　　］に入れるべき適切な字句を解答群から選べ。

［文字列の挿入］

配列 t の任意の場所に，配列 x の文字列を挿入する。文字列 t の文字数は tLen，文字列 x の文字数は xLen に格納されており，文字列 t への挿入位置は pos に格納されている。なお，１≦pos≦tLen＋1 とする。

［文字列の挿入方法］

図３のような状態で文字列を挿入する場合について考える。

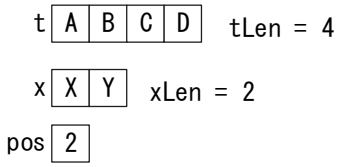

図３　文字列の挿入に関する情報

① 文字列 t の pos で示す位置から，文字列 x の長さを考慮して文字を移動する。図３の例では，配列 t の 2 番目以降を 2 文字分空ける。

図４　文字列 x を挿入する位置を空ける

② 文字列 t の空いた位置へ，文字列 x を転送する。

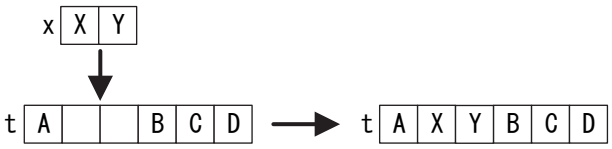

図５　文字列 x を挿入する

［流れ図］

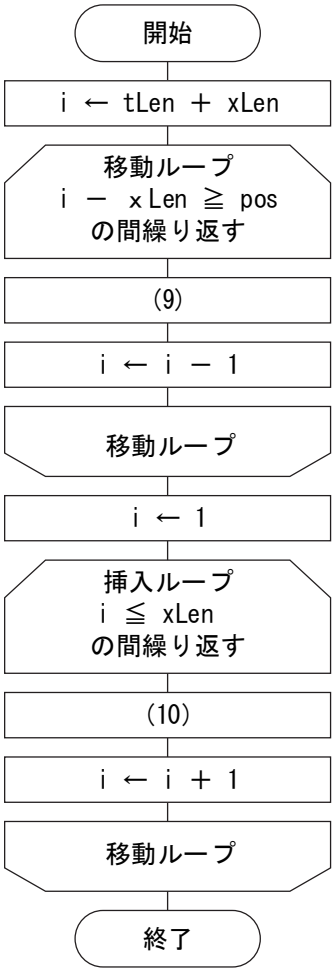

図６　文字列を挿入する流れ図

(9) の解答群

ア. t[i] ← t[i－xLen]　　イ. t[i] ← t[xLen－i]

ウ. t[i－xLen] ← t[i]　　エ. t[xLen－i] ← t[i]

(10) の解答群

ア. t[i] ← x[i＋pos]　　イ. t[i] ← x[i＋pos－1]

ウ. t[i＋pos] ← x[i]　　エ. t[i＋pos－1] ← x[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。

アルゴリズムには様々な種類があり，用途や目的によって利用されるアルゴリズムは異なっている。ここでは，何種類かの簡単なアルゴリズムを確認してみる。

＜設問１＞　次の階乗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］

階乗を再帰的に求めるプログラムFactorialである。例えば5の階乗は5！と表し，

5！＝ 5×4×3×2×1 ＝ 120

となる。

一般形式で表現すると，

ｎ！＝ｎ×(ｎ－1)×(ｎ－2)×・・・×2×1　　（ｎ＝ 0 の場合は 1）

となり，さらに

(ｎ－1)！＝(ｎ－1)×(ｎ－2)×・・・×2×1

を用いて，

ｎ！＝ｎ×(ｎ－1)！　　（ただし，0！＝ 1）

と再帰的に求めることもできる。

［プログラム］

```
○整数型：Factorial ( 整数型：n )
  /* nの階乗を求める */
  if ( n ≧ 1 )
    return [ (1) ]
  else
    return [ (2) ]
  endif
```

(1)，(2) の解答群

ア．0　　イ．1

ウ．Factorial (n - 1)　　エ．Factorial (n + 1)

オ．n×Factorial (n - 1)　　カ．n×Factorial (n + 1)

＜設問２＞　次の配列の要素の和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。なお，配列の添字は 1 から始まる。

［プログラムの説明］

要素数 n の一次元配列の要素のうち，偶数の値を持つ要素の和を求めるプログラム Summary である。例えば，次のような配列では，偶数の値は 6，14，2 となり，求める和は 22 となる。

	1	2	3	4	5
配列 DAT	6	3	14	2	5

［プログラム］

```
○整数型：Summary ( 整数型の配列：DAT, 整数型：n )
  整数型：k, wa
  /* 偶数の値を持つ要素の和を求める */
  wa ← 0
  k ← 0
  while ( k ≦ n )
    if ( [ (3) ] )
      wa ← wa + DAT[k]
    endif
    [ (4) ]
  endwhile
  return wa
```

(3)，(4) の解答群

ア．k ← k - 1　　イ．k ← k + 1

ウ．k ← k + DAT[k]　　エ．(DAT[k] mod 1) ≠ 0

オ．(DAT[k] mod 2) ≠ 0　　カ．(DAT[k] mod 2) = 0

＜設問３＞　次の配列の要素の平均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。なお，配列の添字は１から始まる。

［プログラムの説明］

要素数 n の一次元配列の要素のうち，指定された数値範囲(Low～High)の値の平均値を求めるプログラム Avg である。例えば，次のような配列で Low が 5，High が 15 とすると，数値範囲は 5～15 となり，対象の値は 6，14，5 となり，求める平均値は 8 となる。なお，指定した範囲にはデータが１個以上あるものとし，平均値は整数値で求める(小数点以下切り捨て)。

	1	2	3	4	5
配列 DAT	6	3	14	2	5

［プログラム］

```
○整数型：Avg ( 整数型の配列：DAT, 整数型：n , 整数型：Low, 整数型：High)
  整数型：k, total, count
  /* 対象の値の平均値を求める(小数点以下切り捨て) */
  total ← 0
  count ← 0
  for ( k が 1 から n まで 1 ずつ増やす )
    if ( [ (5) ] )
      total ← total + DAT[k]
      count ← count + 1
    endif
  endfor
  return [ (6) ]
```

(5) の解答群

ア．DAT[k] ≦ Low and DAT[k] ≧ High
イ．DAT[k] ≧ Low and DAT[k] ≦ High
ウ．DAT[k] ≧ Low or DAT[k] ≦ High
エ．DAT[k] ＞ Low or DAT[k] ＜ High
オ．DAT[k] ＞ Low or DAT[k] ＞ High

(6) の解答群

ア．(total - 1) ÷ count　　イ．(total + 1) ÷ count
ウ．total ÷ count　　エ．total ÷ (count - 1)
オ．total ÷ (count + 1)

問題５　次のチェックディジットのアルゴリズムに関する記述を読み，設問に答えよ。

チェックディジットとは，複数の桁により構成されるデータにおいて，入力誤りを検出するためのアルゴリズムである。バーコードや図書コード，クレジットカード番号の管理などで幅広く採用されている。

＜設問＞　次のチェックディジットに関する説明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。

[チェックディジットによる誤り検出]

誤り検出は，各桁のデータ(data)，およびそれに対応する桁の重み(weight)を用意し，データと重みをもとに算出した数値との比較によって行う。例えば，データが[4, 3, 1, 2]，重みが[1, 3, 1, 3]，算出方法がデータの各桁に重みをかけて合計値を取り 11 で割った余りをチェックディジットとする場合，以下のように 9 と求めることができる。

((4 × 1) + (3 × 3) + (1 × 1) + (2 × 3)) mod 11 ＝ 9

チェックディジットが付加された[4, 3, 1, 2, 9]というデータを読み込んだ際，作成時と同じようにチェックディジットを求め，異なっていれば入力データに誤りがあったことが分かる。

なお，本設問のチェックディジットは，求めた数値を対応する文字(図１)に置換しデータに付加する。よって，求めた数値が 9 の場合，チェックディジットは I となる。先の例と同様の算出方法でデータが[2, 5, 4, 1]，桁の重みが[1, 2, 3, 4]の場合，チェックディジットは［(1)］である。データが[4, 1, 3, 2]，桁の重みが[1, 3, 5, 7]，算出方法がデータの各桁に重みをかけて合計値を取り 13 で割った余りをチェックディジットとする場合，チェックディジットは［(2)］となる。

求めた数値	0	1	2	3	4	5	6
置換後の文字	0	A	B	C	D	E	F

求めた数値	7	8	9	10	11	12	…
置換後の文字	G	H	I	J	K	L	…

図１　求めた数値に対応する文字（0～12 までを抜粋）

(1)，(2)の解答群

ア．A　　イ．C　　ウ．E　　エ．F

オ．H　　カ．I　　キ．J

チェックディジットは，メソッド checkDigitCalc()をもつモデルクラス CheckDigit により求める。以下は，CheckDigit 内の checkDigitCalc()について説明する。なお，各データと重みは，各配列の各要素に１つずつ格納されているものとし，配列の添え字は 0 から始める。

[checkDigitCalc()の説明]

チェックディジットの計算メソッドである。データとして以下の要素（図２）からなる data[]，および，その重み weight[]，data[]と weight[]の長さである len を引数として受け取り，戻り値として実行結果のチェックディジットを返す。図 2 のような状態では (3) を返す。

	0	1	2	3	4
配列 data	4	2	3	8	1

	0	1	2	3	4
配列 weight	1	3	1	3	1

図２　引数に受け取る配列の要素

引数とその並び，および戻り値は以下である。

引　数　　整数型の配列： data　　… 各桁のデータ
　　整数型の配列： weight　… データの各桁の重み
　　整数型： len　　… データと重みの長さ
戻り値　　文字型： 文字に置換した後のチェックディジット

［プログラム］

○文字型： checkDigitCalc(整数型の配列： data, 整数型の配列： weight,
整数型： len)

```
文字型の配列：tbl
文字型：checkDigit
整数型：sum, s, i

tbl ← 図１の 0~12 までの数値に対応する文字を順に用意し代入
sum ← 0
s   ← 0
i   ← 0

while( (4) )
    s    ← (5)
    sum  ← sum + s
    i    ← i + 1
endwhile
checkDigit ← tbl[ sum mod 13 ]
return checkDigit
```

(3)の解答群

ア．B　　イ．D　　ウ．G　　エ．K　　オ．L

(4) の解答群

ア．i ≦ len - 2　　イ．i ≦ len - 1

ウ．i ≦ len　　エ．i ≦ len + 1

(5) の解答群

ア．data[i - 1] × weight[i - 1]　　イ．data[i - 1] × weight[i]

ウ．data[i] × weight[i - 1]　　エ．data[i] × weight[i]

＜メモ欄＞

＜メモ欄＞