京都産業大学 コンピュータ理工学部
AO 入試〈情報科目試験型〉

2016 年度入試用 サンプル問題

(2015 年 8 月 27 日版)

*随時、http://info.cse.kyoto-su.ac.jp にてアップデートされる可能性があります。

2016 年度入試（2015 年秋実施）からコンピュータ理工学部の AO 入試には

・〈作品提出型〉

・〈情報科目試験型〉

の２つのタイプが設けられます。

このサンプル問題は、〈情報科目試験型〉でどのような問題を出題するのかを、受験を検討している方に知ってもらう目的で作成しました。実際に行う入試の出題傾向や形式の参考にして下さい。

ただし、次の点には注意して下さい。

(1) 同じ問題は出ません。サンプル問題は、出題傾向や形式を知ってもらう目的で作成しました。この問題を勉強すれば合格するわけではありません。

(2) 設問の内容は高校の「情報」の教科書に書かれている範囲に準拠しています。ただし、入学試験要項にもあるように、論理・思考力、問題解決、応用力、表現力を見ることが目的ですので、必ずしも学校で勉強した通りのことが出題されるとは限りません。

(3) 筆記試験は１次選考として行い、２次選考では面接を行います。筆記試験では、あえて、正答がひとつとは限らない問題を出題し、面接でその解答について説明を求めることも想定しています。志望者の適性や問題解決能力を見ることが目的です。

(4) サンプル問題は I～VI の 6 題を例として挙げていますが、入試当日の問題では 60 分の試験時間を考慮した分量で出題します（3～4 題を予定）。

〔Ⅰ〕 以下の問に答えなさい。

設問（A）、設問（B）の文章について、空欄 A～J を埋める適切な言葉を選択肢の中から選び、記号で答えなさい。

設問（A）

1970 年代に発明された、マルチウィンドウで表現される GUI は［ A ］の略称であり、WIMP インタフェースと表現されることがある。この WIMP の各アルファベットは GUI を構成する要素で、W は［ B ］、I は［ C ］、M は［ D ］、P は［ E ］を意味する。この WIMP インタフェースは、長らくパーソナルコンピュータ向けの画面表示手法と操作手法として利用されてきたが、近年の［ F ］などの登場により、GUI は WIMP で構成されるものではなくなってきた。

A～F の選択肢

ア. ウィンドウ　**イ**. インタフェース　**ウ**. マウス　**エ**. スマートフォン
オ. PDA　**カ**. アイコン　**キ**. メニュー　**ク**. メンテナンス
ケ. プログラミング　**コ**. IoT　**サ**. ポインター　**シ**. キーボード
ス. ナチュラル・ユーザ・インタフェース　**セ**. グラフィカル・ユーザ・インタフェース
ソ. キャラクタ・ユーザ・インタフェース

設問（B）

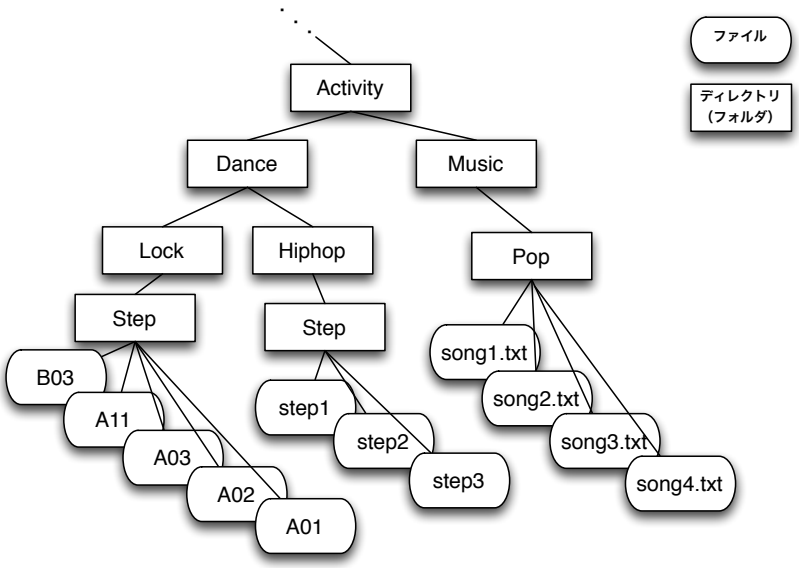
アナログの音やカメラ画像をデジタル化してコンピュータへ取り込む際には［ G ］が行われる。そこでは［ H ］と［ I ］の処理が施され、それらの方法によりコンピュータ上でのデータ量が変わる。例えば、アナログの音を 44100Hz で［ H ］する際に 24bit で［ I ］する場合のデータ量は、16bit で［ I ］する場合に比べて［ J ］倍になる。

G～J の選択肢

タ. D/A 変換　**チ**. A/D 変換　**ツ**. 等価変換　**テ**. 1.5　**ト**. 256　**ナ**. 1024
ニ. 量子化　**ヌ**. 正規化　**ネ**. 標本化　**ノ**. 定常化　**ハ**. 対角化　**ヒ**. 平滑化

〔II〕 次の文章を読んで設問（A）、設問（B）に答えなさい。

ファイルのディレクトリツリー（ファイルのフォルダ階層構造）について考える。次の図で示されるツリー構造で表されるファイルとディレクトリ（フォルダ）があるとする。これは、ある人のダンスや音楽の活動に関する情報を記録したファイルを整理したものである。



これらのうち、B03 という名前のファイルは、本来ならば Dance（ダンス）に関する（属する）ファイルではなく、Music（音楽）に関するファイルであった。そして、それは Pop という音楽ジャンルではなく、Rock という音楽ジャンルを表すディレクトリ（フォルダ）に属するべきファイルであることが判明した。さらに、そのファイル名は B03 ではなく 8beat.txt とすべきであることも判明した。

設問（A）

上記の図で示された誤った状態から、本来の正しいディレクトリ（フォルダ）構成とファイル位置・名前の状態へと変更するには、どのような操作を行えば良いか。次の選択肢（操作）から複数を選択し、適切な操作手順となるように選択肢の記号を並べよ。

選択肢

A. B03 の名前を 8beat.txt に変更
B. Lock 下の Step を Music 下に移動
C. B03 を Music 下の Pop の下へ移動
D. 8beat.txt を Music 下の Pop へ移動
E. Music 下にディレクトリ Rock を作成
F. Pop 下の.txt ファイル全てを削除
G. Dance 下にディレクトリ Rock を作成
H. Hiphop 下の Step の名前を Rock に変更
I. 8Beat.txt の名前を B03 に変更
J. 8beat.txt を Music の下へ移動
K. Pop 下に 8beat.txt を移動
L. 8beat.txt を Music 下の Rock の下へ移動

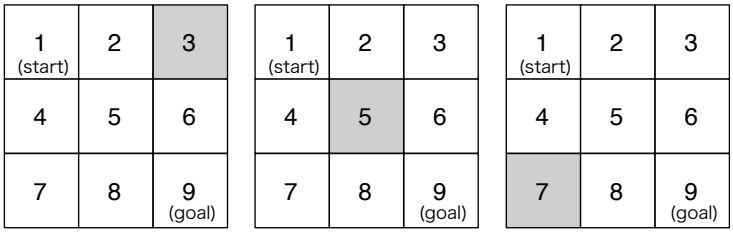
設問（B）

設問（A）で答えた操作手順を行った結果として、本来の正しいファイルとディレクトリツリー（フォルダ階層構造）の状態がどのようなものとなるか、上記の図と同様の描き方で図示せよ。

〔III〕 以下の文章を読んで、設問（A）、設問（B）に答えなさい。

下図で示すような 9 つに区切られたマス目があり、1 のマス目（図中の start）にはロボットが置かれている。この状態からロボットが マス目 9 （goal）まで移動するためのプログラムを作成したい。

ロボットは上下左右に動けるが、斜めには動けない。ただし 3, 5, 7 のマスのいずれか一つはロボットが通行できない状態になっている（つまり盤面状態は、灰色を通行不可の状態として、以下の三つの図のうちのいずれかとする）。事前にどれが通行できないマスなのか知ることはできない。ロボットは自分が隣接する（つまり上下左右の）マス目についてだけ、そこが通行できる状態か通行できない状態か知ることができる。

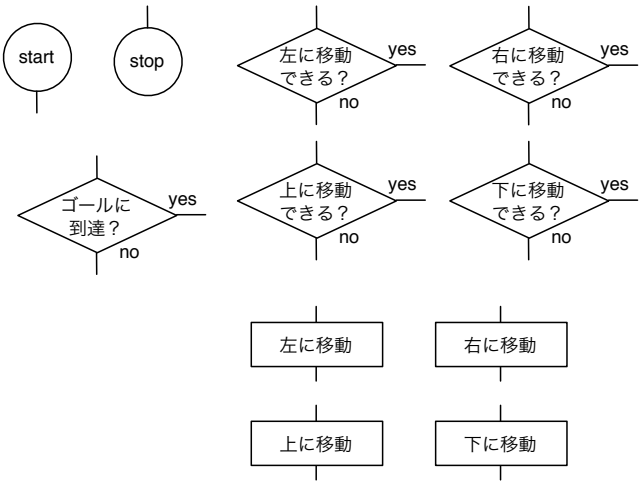


設問（A）

ロボットがスタートの 1 番位置から出発して 9 番のゴールまで、必ず到達できる手順のフローチャートを示せ。フローチャートは右下に示した部品を組み合わせて作成するものとする。

なお、通行できないマス目に衝突する、あるいは 1〜9 までのマスの盤面の外に出てしまうような位置に移動させると、ロボットはエラーとなって停止し、ゴールに到達できなくなる。

フローチャート中の条件判定処理（その方向に移動できるかどうかを調べる）は、指定した方向が移動できるマスである場合にだけ yes と判定するものとする。指定した方向が通行できないマス、あるいは 1〜9 までのマス目以外であった場合は no と判定する。



設問（B）

通行できないマスを 3,5,7 以外の場所に設定した場合、設問（A）で回答したフローチャートのアルゴリズムではロボットがゴールにたどり着けなくなる場合が有り得る。そのような場合の例を最低一つ示し、なぜゴールにたどり着けなくなるかの理由を述べよ。

〔IV〕 以下の文章を読んで、設問（A）～設問（C）に答えなさい。

図 1 に示すように、ブロックが 6 つあり、それぞれ 1 から 6 まで番号がつけられている。これらのブロックは図のように A から F までのラベルが書かれた容れ物に入っている。A～F までの容れ物の隣に、もう一つ、X とラベルが書かれた台が用意されている。

はじめブロックは左から 123456 の順に並んだ状態で容れ物に入っていたが、これを逆順、つまり左から 654321 となるように場所を入れ替えたい。

あなたはロボットアームでブロックをつまんで A-F の容れ物から出して X の台に置いたり、台に置いたブロックを再び A-F の容れ物に戻したりできる。ただし、一つの容れ物には一つのブロックしか入れることはできない。また、ロボットアームが一回につまんで持てるブロックは一つだけとする。複数のロボットアームを用いて複数のブロックを同時に持つといったことも認めない。容れ物から出したブロックを移動できるのは X の台の上か空いている容れ物の中だけであり、台の上には同時に一つのブロックしか置けない。

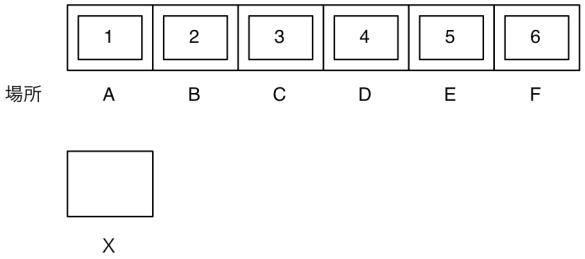


図 1. ブロックの初期状態

設問（A）

以下に、 A の容れ物に入っているブロック 1 と、F の位置にあるブロック 6 の場所を入れ替える作業手順を示す（図 2 参照）。

A → X （A の容れ物の中のブロック(1) をつまんで台 X の上に移動）

F → A （F の容れ物の中のブロック(6) をつまんで A の場所の容れ物の中に移動）

X → F （台 X の上のブロック(1) をつまんで F の場所の容れ物の中に移動）

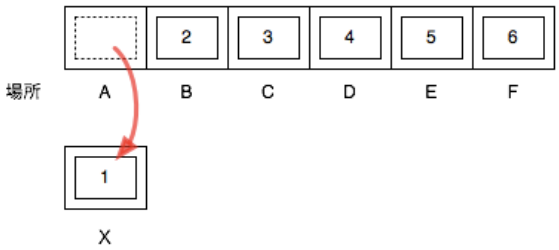


図 2. A → X の操作をしているところ

これで並びは 623451 となった。これまでの作業を「 3 ステップ」と数えることにする。残りの四つのブロックをそれぞれ移動させて 654321 の順に並べ直すために、あと何ステッ

プ必要になるか答えよ。

設問（B）

これまでの問題ではブロックが 6 つであったが、これが 8 つあった場合、同様に 12345678 の並び順から 87654321 の順に並べ直すためには合計で何ステップの作業が必要か。必要なステップ数と、その理由を答えよ。

設問（C）

123456 と並んでいる 6 つのブロックを完全に逆順にするのではなく、その前半（123）と後半（456）のブロックの位置を入れ替えるだけで良い、とした場合、何ステップの作業が必要か（例えば 456123 の順でも良い）。必要なステップ数と、その理由を答えよ。

〔V〕 以下の文章を読んで、設問（A)、設問（B）に答えなさい。

画面上の指定したピクセル(画素)の色を変化させられるプログラミング環境があるとする。図 1 のように、画面の左右方向に X 軸、上下方向に Y 軸をとり、画素の位置（X, Y）は左下を原点（0, 0）として表現する。X や Y の値が 1 増えると、画素の位置は一つ右や上となる。

あなたは指定した位置の画素の色を黒く変化させることができる。この作業を「打点する」と呼ぶ。つまり「(100, 200) の位置に打点する」などと命令すると、位置（100, 200）の画素が黒くなる。X, Y 軸とも位置は正の整数のみで指定し、実数で与えた（小数点以下の値があった）としても小数点以下の数値は切り落として処理される。

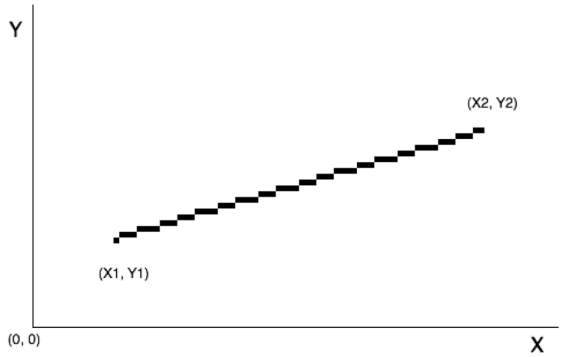


図 1. (X1, Y1) から (X2, Y2) まで斜めの線を描画した状態

設問（A）

この環境で図 1 のように位置 (X1, Y1) から (X2, Y2) にめがけて斜めの線を描画したい。

そのような結果が得られるプログラム（処理手順）を 図 2 にフローチャートで示した。

手順を以下に解説する。

・変数 AX を X1 から X2 まで 1 ずつ変化させてループし、繰り返しの度に、
・描画処理全体のどのくらいまで進んだかを計算（処理 1 に相当）して変数 R に格納し
・その値を用いて打点すべき Y 座標の値を求めて（処理 2 に相当）変数 AY に格納し
・位置 (AX, AY) に打点する

フローチャート中の「処理 1」「処理 2」で実行すべき具体的な計算式を答えよ。

なお X1 と X2 が同じ値になることはなく、Y1 と Y2 が同じ値になることもない。また、X1, Y1, X2, Y2 はすべて正の整数であり、X1 < X2, Y1 < Y2 である。図 1 の座標軸は説明のために描いたものであり、このプログラムで座標軸を描くことはない。演算は実数で行われ、各変数は実数を格納できることとする。打点処理において小数点以下は切り落として処理される。

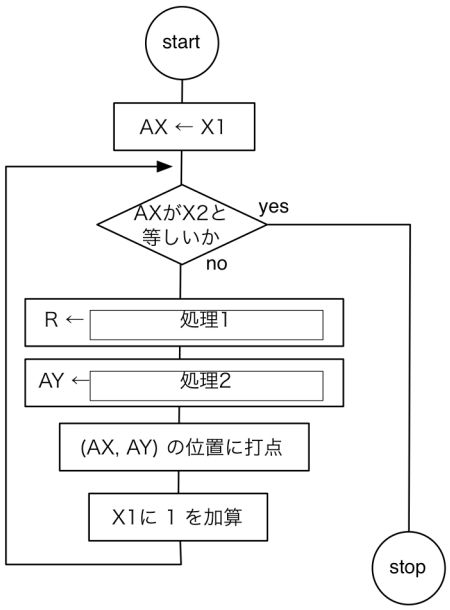


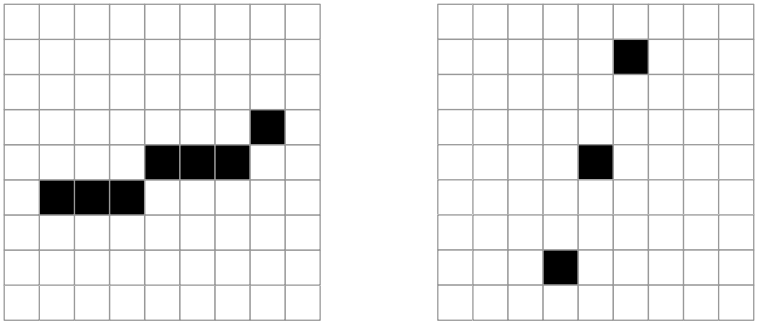
図 2. 斜めの線を描画する処理

設問（B）

この描画手順では、斜めの線の角度が 45 度あるいはそれより（横に）寝ている場合、つまり水平を 0 度として 0 度より大きく 45 度以下の範囲であれば打点した画素は、すぐ上下左右あるいは斜めの隣接した画素と連続して描画される（図 3 の左側の状態）。

しかし描画する線の角度が 45 度より（縦に）立っている場合、打点した画素が離れてしまう（図 3 の右側の状態）。

このような、ドットが離れてしまう問題を解決する（線が縦に立っている場合でもドットが離れて打点されない）ためには、プログラム（処理手順）にどのような改善を加えれば良いか述べよ。



ドットが隣接している場合　　ドットが離れてしまった場合

図 3.　描画した結果（ドットが隣接している状態と離れている状態）

〔VI〕 以下の文章を読んで、設問（A)、設問（B）に答えなさい。

選果場のベルトコンベアで、ピーマンが次々に運ばれてくる。ピーマンは１つずつ自動的に秤（はかり）に乗せられて重さを計測される。基準より重いピーマンは袋に詰めて出荷され、そうでないものは出荷せずに取り除かれる。このような動作を制御するためにコンピュータが接続されている。

コンピュータのプログラムで使用可能な手続きは以下に示す通りである。

`check()`　１つのピーマンが秤に乗るのを待ってその重さを量り、返り値としてピーマンの重さの計量結果（グラム単位）を返す。手続きに引数はない。

`drop()`　秤の皿を傾けて、上に乗ったピーマンを下ろす。引数として 0 を指定すると、ピーマンは出荷用の袋に入れられる。引数として 1 を指定すると、ピーマンは袋には入れられずに取り除かれ、出荷されない。手続きの返り値はない。

`pack()`　ピーマンの入った袋の口を閉じて出荷に回し、次の袋の用意をする。手続きに引数、および返り値はない。

プログラムでは式として、整数定数、変数、手続きの返り値、およびこれらに対して加減演算 (+, -) を適用したものが記述できる。また、表１に示す比較演算子で２つの式を比較して真偽値を返すことができ、これを条件式と呼ぶ。比較演算子の種類と意味は表１に示した通りである。さらに条件式として、真を表す定数 1 を記述できる。

表 1. 条件式で利用できる演算子

a == *b*	*a* と *b* が等しい
a != *b*	*a* と *b* が等しくない
a > *b*	*a* が *b* より大きい
a >= *b*	*a* が *b* 以上
a < *b*	*a* が *b* より小さい
a <= *b*	*a* が *b* 以下

プログラムは以下の構文を使って記述する。

変数 ← 式　変数に式の値を代入する。A, B, C, D という名前の４つの変数を任意の目的で使用できる。

```
if 条件式
   ...
end
```
条件式が真の時、end までの文を実行する。条件式が偽の時は何も実行しない。

```
if 条件式
   ...
else
   ...
end
```
条件式が真の時、else までの文を実行し、偽の時は else から end までの文を実行する。

```
while 条件式
   ...
end
```
条件式が真である間、end までの文を繰り返し実行する。

例として、重さに関係なく、袋に 10 個のピーマンを詰めて出荷するプログラムをリスト 1 に示す。

設問（A）

袋に入れたピーマンの合計の重さが 300 グラム以上になったら袋の口を閉じるようなプログラムを作成したい。リスト 2 の空欄(a), (b)を埋めよ。

設問（B）

リスト 2 のプログラムを修正して、30 グラムより軽いピーマンは出荷せずに取り除き、一袋が 300 グラム以上になるようにしたい。そのようなプログラムを記述せよ。

```
while 1
  A ← 0
  while A < 10
    B ← check()
    drop(0)
    A ← A + 1
  end
  pack()
end
```

リスト 1

```
while 1
  A ← 0
  while [ (a) ]
    A ← [ (b) ]
    drop(0)
  end
  pack()
end
```

リスト 2

〔解答例〕

〔I〕

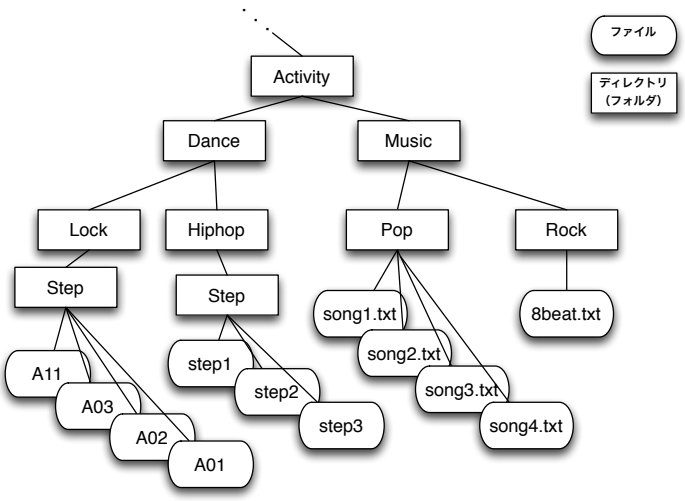
A: セ　　B: ア　　C: カ　　D: キ　　E: サ　　F: エ
G: チ　　H: ネ　　I: ニ　　J: テ

〔II〕

設問A

操作手順の記号列：　A→E→L　もしくは　E→A→L

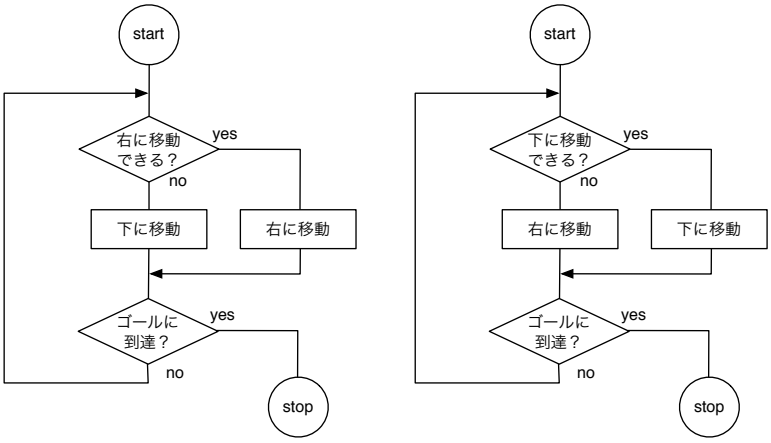
設問B



〔III〕

設問A

２つの解答例を示す。どちらも正解である。



設問B

注意：設問Aに回答したアルゴリズムが異なるものであった場合、当然この回答も違ったものになります。設問Aの回答と合わせて妥当な回答であれば正解として評価します。ここでは回答例の左側のアルゴリズム（右に進めるかどうかをチェックする）に対応する回答例を示します。評価の対象は受験者がアルゴリズムを正しく把握して論理的に動作を説明できることを中心に見ますので、説明ロジックが回答例の通りでなくても構いません。

通行できないマスを 6 に設定すると、設問Aで回答したアルゴリズムではゴールにたどり着けなくなる。

理由： ロボットは マス 1 から出発して右が空いているために マス 2, 3 へと進む。その後に「右に移動できる」かどうか判定するが、(1-9 のマス目の範囲から外に出てしまうために) 判定結果は no となり、下に移動しようとする。しかし、下つまりマス 6 は「通行できないマス」であるため、そこでエラーとなって停止してしまう。

〔IV〕

設問A 6 ステップ

設問B 12 ステップ

理由：4ペアそれぞれを個別に入れ替えるのに 3 ステップずつ必要であるから

設問C 7 ステップ

理由：たとえば、A → X, D → A, B → D, E → B, C → E, F → C, X → F の手順で、ブロックは 456231 の順に並ぶ。これは、最初に一つ空いている容れ物を作りさえすれば、あとはすべて 1 ステップで目標位置に移動させることができるためである。つまり、最初に一つ、台に「よける」作業が余分になるだけで、それ以外の無駄は無い。したがって、 N ブロックあるのなら N+1 回の移動で良い。

〔V〕

設問A

処理 1. (AX - X1) / (X2 - X1)

処理 2. (Y2 - Y1) * R + Y1

設問B

回答例：その一

角度を調べて縦に立っている場合と横に寝ている場合で処理を分けてしまうことで対処できる。

まず、 AX を 1 ずつ変化させて AY を求める図 2 の処理の手順を真似て、AY を 1 ずつ変化させて AX を求める手順を用意する。

処理全体としては、まず描画する線の角度を調べて、縦に立っている場合は AY を変化させて AX を求める処理に分岐し、そうでなかった場合は AX を変化させて AY を求める処理に分岐すれば良い。

線の角度の判定は、 条件式 (X2 - X1) > (Y2 - Y1) の結果が真であれば横に寝ている、とすれば良い。

回答例：その二

打点に隙間が空かないよう、充分小さい値（例えば 0.001）ずつ AX を変化させて AY を求めることで対処できる。

それだけでは必要以上に打点してしまい効率が悪いが、最後に打点した画素の位置（整数に切り落とした値）を別の変数、たとえば KX, KY に保存しておき、求めた新しい打点位置がこれと同じになるならば打点処理を行わない（処理をスキップする）ようにするなどの工夫もできる。

補足：

その一の回答例の方が処理的に効率が良い（無駄な処理を行わない）点から、その二の回答例より高く評価します。

上記の回答例は幾らか精密さに欠けるところがあります。その一の例では実数演算の誤差によって隙間が生じる場合、その二の例でもやはり隙間が空いてしまう場合があり得ます。そのような精密な理解については面接で質問・確認して、（面接での）評価に加える場合もあります。

〔VI〕

設問A

(a) A < 300

(b) A + check( )

設問B

```
while 1
  A ← 0
  while A < 300
    B ← check()
    if B >= 30
      A ← A + B
      drop(0)
    else
      drop(1)
    end
  end
  pack()
end
```