平成 18 年度　春期

基本情報技術者
午後　問題

注意事項

1. 試験開始の合図があるまで，問題冊子を開いて中を見てはいけません。
2. この注意事項は，問題冊子の**裏表紙**に続きます。問題冊子を裏返して必ず読んでください。
3. **答案用紙への受験番号などの記入及びマークは，試験開始の合図があってから始めてください。**
4. 試験時間は，次の表のとおりです。

試験時間	13:00 ～ 15:30（2 時間 30 分）

途中で退出する場合には，手を挙げて監督員に合図し，答案用紙が回収されてから静かに退出してください。

退出可能時間	13:40 ～ 15:20

5. 問題は，次の表に従って解答してください。

問題番号	問 1 ～ 問 5	問 6 ～ 問 9	問 10 ～ 問 13
選択方法	全問必須	1 問選択	1 問選択

選択した問題については，答案用紙の選択欄の (選) をマークしてください。マークがない場合は，採点の対象になりません。

6. **問題に関する質問にはお答えできません。**文意どおり解釈してください。
7. 問題冊子の余白などは，適宜利用して構いません。
8. アセンブラ言語の仕様は，この冊子の末尾を参照してください。
9. 電卓は，使用できません。

**注意事項は問題冊子の裏表紙に続きます。
こちら側から裏返して，必ず読んでください。**

正　誤　表

平成18年4月16日実施

基本情報技術者試験　午後　問題

ページ	問題番号	行	誤	正	訂正の内容
17	4	上から12行目	`Idx ← ClstS`	`・Idx ← ClstS`	下線部分を追加する。

共通に使用される擬似言語の記述形式

擬似言語を使用した問題では，各問題文中に注記がない限り，次の記述形式が適用されているものとする。

〔宣言，注釈及び処理〕

記述形式		説明
○		手続，変数などの名前，型などを宣言する。
/* 文 */		**文**に注釈を記述する。
処理	・**変数** ← **式**	**変数**に**式**の値を代入する。
	・**手続**(**引数**, …)	**手続**を呼び出し，**引数**を受け渡す。
	▲ **条件式** 　**処理** ▼	単岐選択処理を示す。 **条件式**が真のときは**処理**を実行する。
	▲ **条件式** 　**処理 1** —— 　**処理 2** ▼	双岐選択処理を示す。 **条件式**が真のときは**処理 1** を実行し，偽のときは**処理 2** を実行する。
	■ **条件式** 　**処理** ■	前判定繰返し処理を示す。 **条件式**が真の間，**処理**を繰り返し実行する。
	■ 　**処理** ■ **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**を繰り返し実行する。
	■ **変数**: **初期値**, **条件式**, **増分** 　**処理** ■	繰返し処理を示す。 開始時点で**変数**に**初期値**（定数又は式で与えられる）が格納され，**条件式**が真の間，**処理**を繰り返す。また，繰り返すごとに，**変数**に**増分**（定数又は式で与えられる）を加える。

〔演算子と優先順位〕

演算の種類	演算子	優先順位
単項演算	+，−，not	高 ↑
乗除演算	×，÷，%	
加減演算	+，−	
関係演算	＞，＜，≧，≦，＝，≠	
論理積	and	
論理和	or	↓ 低

注　整数同士の除算では，整数の商を結果として返す。%演算子は，剰余算を表す。

〔論理型の定数〕

true，false

次の問1から問5までの5問については，全問解答してください。

問1　関係データベースに関する次の記述を読んで，設問1，2に答えよ。

ある会社には，次の社員表，家族表からなる関係データベースがある。

社員表

社員ID	氏名	年齢
S035	浅井健司	34
S612	上野佳子	23
S052	佐藤剛	23
S054	鈴木太郎	22
S042	田中順	31
S609	中村和子	25
S048	渡辺正	26

家族表

社員ID	家族氏名	年齢	続柄
S035	浅井一	70	父
S035	浅井裕子	65	母
S035	浅井美智子	34	妻
S035	浅井あずさ	11	子
S035	浅井大	5	子
S612	上野博	55	父
S612	上野春子	53	母
S612	上野恵美	27	姉
S054	鈴木淳子	23	妻
S054	鈴木翔	1	子
S042	田中奈津子	30	妻
S042	田中雄太	4	子
S609	中村浩	60	父
S609	中村悦子	52	母

設問1　SQL文①とSQL文②を実行した結果について，次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

SQL文①

```
SELECT 社員ID FROM 社員表
  WHERE 年齢 > (SELECT AVG(年齢) FROM 家族表) ORDER BY 社員ID
```

SQL文②

```
SELECT 社員ID FROM 社員表
  WHERE 年齢 > (SELECT AVG(年齢) FROM 家族表
    WHERE 家族表.社員ID = 社員表.社員ID) ORDER BY 社員ID
```

SQL文①を実行した結果は［ a ］となる。

SQL文②を実行した結果は［ b ］となる。

解答群

ア

S042
S048
S052
S054

イ

S042
S054

ウ

S048
S052
S609
S612

エ

S054

オ

S609
S612

カ

問合せ結果0件

設問2 SQL文③と同じ実行結果が得られると考え，SQL文④を実行したところ，異なる結果が得られた。その理由を説明する次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

SQL文③

```
SELECT 社員ID FROM 社員表
  WHERE 年齢 < (SELECT MIN(年齢) FROM 家族表
    WHERE 家族表.社員ID = 社員表.社員ID)
```

SQL文④

```
SELECT 社員ID FROM 社員表
  WHERE 年齢 < ALL (SELECT 年齢 FROM 家族表
    WHERE 家族表.社員ID = 社員表.社員ID)
```

SQL文④の限定子ALLを使った限定比較の場合には，［ c ］社員も抽出されるが，SQL文③のように集合関数と比較述語を使った場合は，［ c ］社員は抽出されない。つまり，SQL文④の実行結果には，次のSQL文⑤によって抽出される社員も併せて抽出されることになる。

SQL文⑤

```
SELECT 社員ID FROM 社員表 WHERE [ d ]
```

cに関する解答群

ア　家族がいない　　イ　家族がいる

dに関する解答群

ア　EXISTS (SELECT * FROM 家族表)

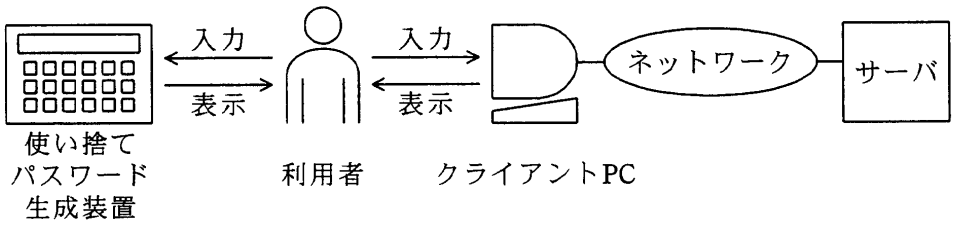
イ　NOT EXISTS (SELECT * FROM 家族表)

ウ　EXISTS
(SELECT * FROM 家族表 WHERE 家族表.社員ID = 社員表.社員ID)

エ　NOT EXISTS
(SELECT * FROM 家族表 WHERE 家族表.社員ID = 社員表.社員ID)

問2　サーバへのログイン管理に関する次の記述を読んで，設問に答えよ。

使い捨てパスワード（One-Time Password : OTP）の仕組みを応用して作られた，ログイン管理システムである。



〔ログイン管理システムの説明〕

サーバへのログイン可能回数 M と定数 K を決定し，M 個の有効な使い捨てパスワードを使用する。残りのログイン可能回数が n のときの使い捨てパスワード otp(n) には，一方向性関数 hash を用いて，次式で得られる値を用いる。

$$otp(n) = \overbrace{hash(hash(\cdots hash(K)\cdots))}^{n\text{回}}$$

(1) 使い捨てパスワード生成装置

利用者が，使い捨てパスワードをすべて記憶し，管理することは困難なので，定数 K と残りのログイン可能回数 n から使い捨てパスワードを生成し，表示する携帯式の使い捨てパスワード生成装置を用いる。

第 1 回目のパスワード生成時は n ＝ M で，パスワードを生成するたびに n を 1 ずつ減らし，最後のパスワード生成時は，n ＝ 1 となる。

(2) 利用者

利用者は，K を使い捨てパスワード生成装置に入力し，表示された使い捨てパスワードを，サーバへのログインパスワードとしてクライアント PC に入力する。

(3) サーバ

サーバは，次の二つを保持する。

① 使い捨てパスワード生成装置と同じ一方向性関数 hash

② パスワード検査に用いるために，利用者が，直前の許可されたログインで使ったパスワード otp(n＋1)

サーバでのパスワード検査は，ログイン時にクライアント PC からサーバへ送られてきた使い捨てパスワード otp(n) に，hash を 1 回適用し，hash(otp(n)) を得て，それがサーバの保持している otp(n+1) と一致するかどうかで行う。一致すれば，サーバは，ログインを許可する。

サーバは，保持している otp(n+1) を次回の検査用に更新する必要がある。このためには，クライアント PC から送られてきた使い捨てパスワードの値 otp(n) で置き換えればよい。

図にログイン管理の流れを示す。ここでは，サーバには，利用者の otp(n) を受け入れる準備として，otp(n+1) が保持されているものとする。また，この図では，時間は上から下に向かって進むように表現されている。

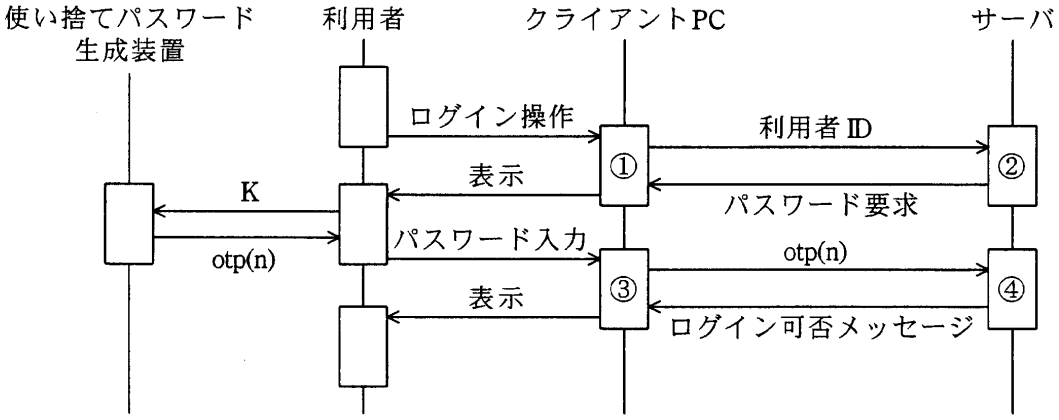


図　ログイン管理の流れ

〔図中の主な処理の説明〕

① 利用者 ID によってサーバにログイン操作を行う。

② パスワードを要求する。

③ 使い捨てパスワード otp(n) をサーバへ送る。

④ 受け取った otp(n) に hash を適用し，hash(otp(n)) を得る。利用者 ID ごとに保存していた otp(n+1) と比較する。一致すれば，ログインを許可し，受け取った otp(n) を次回のパスワード検査用に保持する。一致しなければ，ログインを拒否する。

設問　このログイン管理に関する次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

利用者がパスワードとしてクライアントPCに入力する［ a ］は，ネットワーク上の通信メッセージにも含まれる。しかし，［ a ］は，ログイン許可と同時に無効なパスワードとなるので，これを盗聴してその後で使用しても，サーバにログインすることはできない。

サーバが保持する使い捨てパスワードotp(n＋1)は，使用済みの無効なパスワードなので，これを不正に入手して使用しても，サーバにログインすることはできない。また，otp(n＋1)とhashを不正に入手したとしても，hashの一方向性の特徴から，これらを基に，未使用の使い捨てパスワードを得ることは極めて困難である。

このログイン管理の仕組みで，不正アクセスの危険性が大きいのは，［ b ］と［ c ］の両方が不正に入手，使用された場合である。

このうち，［ b ］は，サーバと使い捨てパスワード生成装置にある。サーバには，セキュリティ強化のための既存の手段を講じることができるが，使い捨てパスワード生成装置は，別の利用者も所有しているので，その不正使用及び盗難の危険性は比較的高い。

［ c ］は，クライアントPC，ネットワーク上の通信メッセージ及びサーバには一時的にも存在しない情報なので，これらから盗まれる危険性はない。また，［ c ］は，利用者本人だけが知る秘密情報である。

解答群

ア　hash	イ　K	ウ　M
エ　n	オ　otp(M＋1)	カ　otp(n)
キ　利用者ID		

問3　通信ネットワークの信頼性に関する次の記述を読んで，設問1，2に答えよ。

三つの拠点A，B，Cが図1のように3本の回線で接続されている。各回線の稼働率と通信速度は図1のとおりである。拠点Aと拠点Cとの間で業務処理の通信が行われており，これらの回線はそのためだけに利用されている。この業務処理では，週末には50 Mbps，月末には100 Mbps，それ以外の通常時には40 Mbpsの通信速度を確保する必要がある。

図1の拠点Bと拠点Cとの間のように拠点間に複数の回線が存在する場合，その間の通信速度は各回線の通信速度の和になるものとする。また，1本の回線に障害が発生しても残りの回線で通信が可能であり，このときの回線切替えに伴う通信遅延などは考慮しなくてよい。

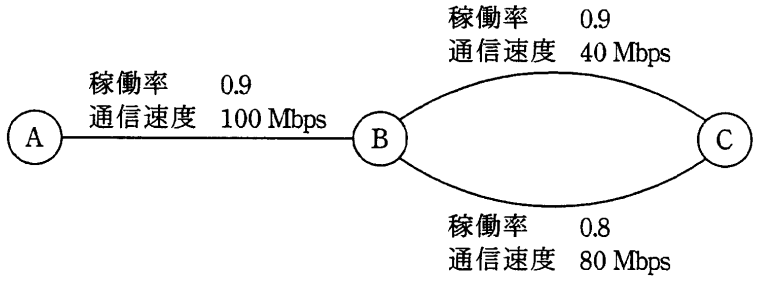


図1　拠点A，B，C間のネットワーク接続状況

設問1　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

拠点Aと拠点Cとの間で，通常時に必要な通信速度が確保できるネットワークの稼働率は［　a　］，週末に必要な通信速度が確保できるネットワークの稼働率は［　b　］である。

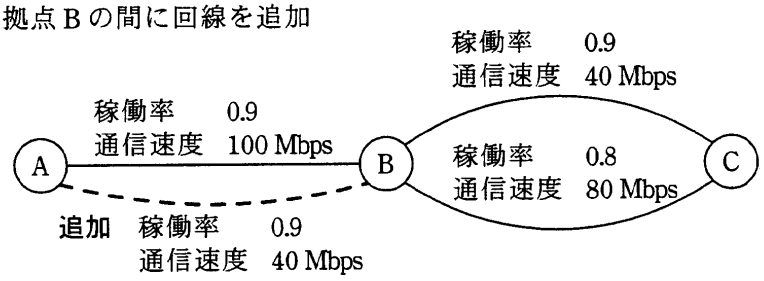
解答群

ア　0.648	イ　0.72	ウ　0.8
エ　0.81	オ　0.882	カ　0.9

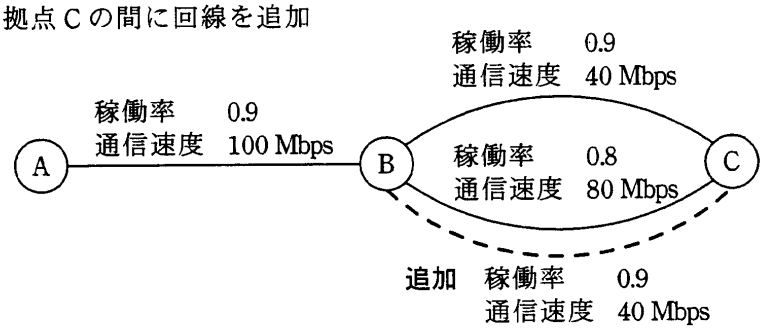
設問2　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

拠点Aと拠点Cとの間のネットワークの稼働率を高めるために，新たに稼働率0.9，通信速度40 Mbpsの回線を追加することにした。回線の追加方法として，図2に示す3通りを検討している。稼働率は，通常時，週末，月末のすべての場合に対して高めたいと考えている。この条件を満たす方法は，［　c　］である。さらに，通常時，週末，月末のすべての場合の稼働率を高めるだけでなく，月末の処理の増大に備え，回線に障害が発生していないときの最大通信速度も上げるために選択すべき方法は，［　d　］である。

(1) 拠点Aと拠点Bの間に回線を追加



(2) 拠点Bと拠点Cの間に回線を追加



(3) 拠点Aと拠点Cを直結する回線を追加

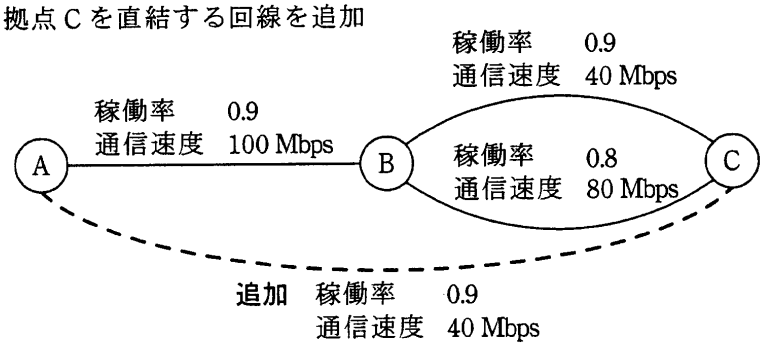


図2　回線の3通りの追加方法

解答群

ア (1) だけ	イ (2) だけ	ウ (3) だけ
エ (1) 又は (2)	オ (1) 又は (3)	カ (2) 又は (3)

問4　次のプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラム1の説明〕

関数 CreateFile は，ディスクにファイルの領域を確保するプログラムである。

(1) ディスク上のファイルは，ファイル管理テーブル（以下，FCT という）とファイル割当テーブル（以下，FAT という）で管理される。

なお，ファイル名は一意である。

(2) ディスクは，クラスタと呼ばれる一定の大きさの領域に区切られる。クラスタには，1から始まる番号が順に付けられている。ファイルは，1個以上のクラスタを使用する。複数のクラスタを使用するファイルは，クラスタを連結リストとして使うので，必ずしも連続するクラスタを使用するとは限らない。

なお，クラスタの大きさは1,024バイトである。

(3) FCT は構造体の配列であり，構造体のメンバとその意味は次のとおりである。

```
構造体型: FCT[1000] {
  文字型: Name,          /* ファイル名 */
  日時型: Cdate,         /* 作成日時 */
  整数型: Size,          /* ファイルの大きさ */
  整数型: ClstS          /* 先頭クラスタの番号 */
}
```

例えば，第10要素のメンバ Name の参照は，FCT[10].Name と表記する。

(4) FAT は，クラスタの使用状況を示す整数型の配列であり，クラスタの状態に応じて，次に示す値が格納されている。

i番目のクラスタの状態		FAT[i]の値
未使用		0
使用中	ファイルの末尾以外	次のクラスタの番号
	ファイルの末尾	−1

(5) FCT と FAT は，ともに大域変数であり，配列の要素番号は1から始まる。

(6) FCT と FAT の使用例を，次に示す。ここで，Name の内容が空文字列の場合は，その配列要素が未使用であることを示す。

FCT

	ファイル名 (Name)	作成日時 (Cdate)	ファイルの大きさ (Size)	先頭クラスタの番号 (ClstS)
1	AAA	2006-01-04 10:01	1500	1
2	(空文字列)			
3	BBB	2006-01-05 14:21	800	5
4	CCC	2006-01-05 20:45	2500	4
5	(空文字列)			
	⋮	⋮	⋮	⋮

FAT

1	2
2	−1
3	0
4	6
5	−1
6	7
7	−1
8	0
	⋮

(7) 関数 CreateFile がファイルの領域を確保する手順は，次のとおりである。

① FAT 内を先頭から順番に調べて，引数で指定されたファイルが格納できる必要最小限の個数のクラスタを確保する。

② FCT 内を先頭から順番に調べて，未使用要素（メンバ Name の内容が空文字列）に情報を記録する。

③ FAT 内で必要なクラスタが確保できない場合や，FCT 内に未使用要素がない場合はエラーとする。

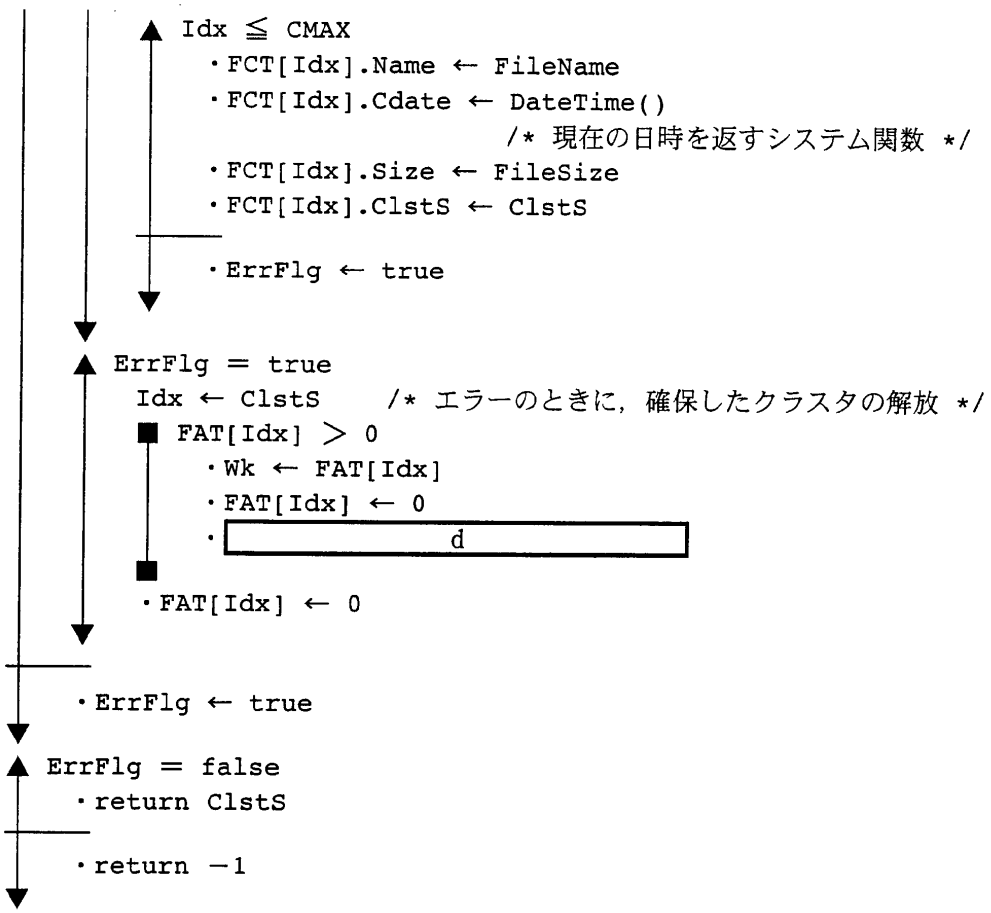
(8) 関数 CreateFile の引数と返却値の仕様を次に示す。

引数名／返却値	データ型	入力／出力	意味
FileName	文字型	入力	作成するファイル名
FileSize	整数型	入力	作成するファイルの大きさ（バイト数） FileSize ＞ 0
返却値	整数型	出力	1 以上のとき先頭クラスタの番号 −1 のときエラー

〔プログラム１〕

```
○大域:整数型: CSIZE = 1024          /* クラスタの大きさ */
○大域:整数型: AMAX = 2000           /* FATの要素数 */
○大域:整数型: CMAX = 1000           /* FCTの要素数 */
○大域:整数型: FAT[AMAX]
○大域:構造体型: FCT[CMAX] {
        文字型: Name,                /* ファイル名 */
        日時型: Cdate,               /* 作成日時 */
        整数型: Size,                /* ファイルの大きさ */
        整数型: ClstS                /* 先頭クラスタの番号 */
      }

○整数型: CreateFile(文字型: FileName, 整数型: FileSize)
○整数型: ClstNum, ClstS, Idx, Wk
○論理型: ErrFlg
・ErrFlg ← false
・ClstNum ← [          a          ]
・Idx ← 1
■ Idx ≦ AMAX and FAT[Idx] ≠ 0
    ・Idx ← Idx + 1
■
▲ Idx ≦ AMAX
    ・ClstS ← Idx
    ・ClstNum ← ClstNum − 1          /* クラスタの確保 */
    ■ ClstNum > 0 and ErrFlg = false
        ・Wk ← Idx
        ・Idx ← Idx + 1
        ■ Idx ≦ AMAX and FAT[Idx] ≠ 0
            ・Idx ← Idx + 1
        ■
        ▲ Idx ≦ AMAX
            ・[          b          ]
            ・ClstNum ← ClstNum − 1
        ─────
            ・ErrFlg ← true
    ■
    ▲ ErrFlg = false
        ・[          c          ]
        ・Idx ← 1                    /* FCT の未使用要素に記録 */
        ■ Idx ≦ CMAX and FCT[Idx].Name ≠ 空文字列
            ・Idx ← Idx + 1
        ■
```



設問1 プログラム1中の［　　］に入れる正しい答えを，解答群の中から選べ。

aに関する解答群

ア FileSize ÷ CSIZE

イ FileSize ÷ CSIZE + 1

ウ FileSize ÷ CSIZE − 1

エ (FileSize + CSIZE + 1) ÷ CSIZE

オ (FileSize + CSIZE − 1) ÷ CSIZE

b ～ d に関する解答群

ア　FAT[Idx] ← Idx　　イ　FAT[Idx] ← Wk

ウ　FAT[Idx] ← −1　　エ　FAT[Wk] ← Idx

オ　FAT[Wk] ← Wk　　カ　FAT[Wk] ← −1

キ　Idx ← FAT[Idx]　　ク　Idx ← FAT[Wk]

ケ　Idx ← Wk

設問2　関数 DeleteFile は，ファイルを削除するプログラムである。次のプログラム2中の［　　］に入れる正しい答えを，解答群の中から選べ。ただし，［ d ］には設問1の正しい答えが入っているものとする。

関数 DeleteFile の引数と返却値の仕様を次に示す。

引数名／返却値	データ型	入力／出力	意味
FileName	文字型	入力	削除するファイル名
返却値	整数型	出力	0のとき正常終了 −1のときエラー

〔プログラム2〕

```
○整数型: DeleteFile(文字型: FileName)
○整数型: Idx, Wk
・Idx ← 1
［　　e　　］
  ・Idx ← Idx + 1

［　　f　　］
  ・FCT[Idx].Name ← 空文字列
  ・Idx ← FCT[Idx].ClstS          /* クラスタの解放 */
  ■ FAT[Idx] ≠ −1
    ・Wk ← FAT[Idx]
    ・FAT[Idx] ← 0
    ・［　　d　　］
  ・FAT[Idx] ← 0
  ・return 0
  ・return −1
```

解答群

ア　Idx ≦ CMAX

イ　Idx ＞ CMAX

ウ　Idx ≦ CMAX and FCT[Idx].Name ≠ FileName

エ　Idx ≦ CMAX or FCT[Idx].Name ≠ FileName

オ　Idx ＞ CMAX and FCT[Idx].Name ≠ FileName

カ　Idx ＞ CMAX or FCT[Idx].Name ≠ FileName

問5　プログラム設計に関する次の記述を読んで，設問1～3に答えよ。

ある通信販売会社では，期間限定のキャンペーンを実施する。このキャンペーンに応募し，期間中の1回の購入金額が3,000円以上であった顧客に，購入回数に関係なく賞品抽選券を1枚だけ郵送することにした。このために，既存のファイルから対象となる顧客を選び，当該顧客のレコードをあて先ファイルに出力するプログラムを設計する。

〔ファイルの説明〕

(1) 売上伝票ファイルのレコードは，1回の購入ごとに1件作成されている。

(2) 売上伝票ファイルは，購入日付をキーとして，昇順に整列されている。

(3) 顧客番号は，6けたの数字で構成されている。

(4) キャンペーンに応募した顧客の顧客番号は，応募者ファイルに登録されている。応募者ファイルには，同じ顧客番号のレコードが複数存在することはない。

(5) 応募者ファイルは，顧客番号をキーとして，昇順に整列されている。

(6) 顧客情報は，顧客マスタファイルに登録されている。

(7) 売上伝票ファイル，応募者ファイル，顧客マスタファイル及びあて先ファイルの様式を図1に示す。

売上伝票ファイル

購入日付	顧客番号	商品コード	購入数量	購入金額

応募者ファイル

顧客番号

顧客マスタファイル

顧客番号	顧客住所	顧客氏名

あて先ファイル

顧客番号	顧客住所	顧客氏名

図1　各ファイルの様式

入力ファイルと出力ファイルの様式を検討した結果，図2に示すように，売上伝票ファイルに対する前処理プログラムとあて先ファイル作成プログラムの二つに分割する設計とした。

このうち，あて先ファイル作成プログラムは，中間ファイルと応募者ファイルから

レコードを順番に読み，突合せ処理によって，両方のファイルに存在する顧客番号について，顧客情報を顧客マスタファイルから得て，あて先ファイルを作成する。

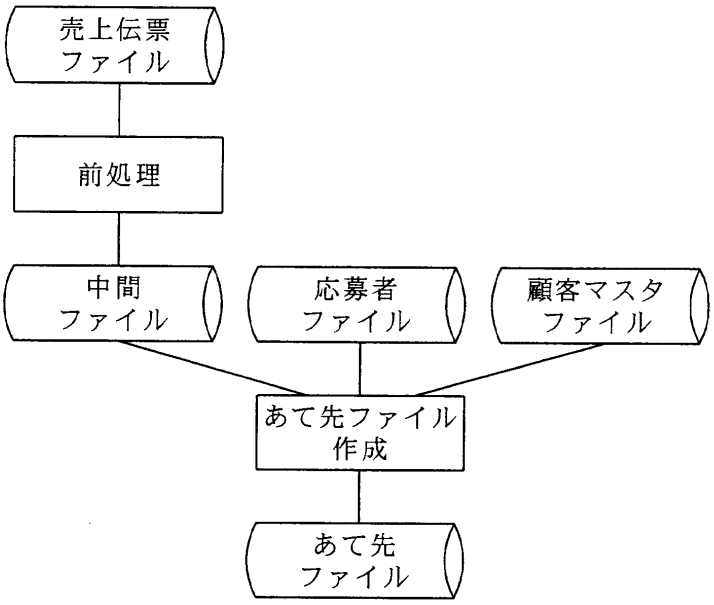


図2　分割した処理と入出力ファイル

設問1　前処理プログラムが行う処理のうち，購入金額が3,000円以上であるレコードの抽出以外の処理として，正しい答えを，解答群の中から二つ選べ。

なお，前処理プログラムの出力である中間ファイルの様式は，売上伝票ファイルと同じとする。

解答群

ア　応募者のレコードの抽出

イ　キャンペーン対象期間のレコードの抽出

ウ　購入金額をキーとする整列

エ　購入数量が正数であるレコードの抽出

オ　購入日付をキーとする整列

カ　顧客番号をキーとする整列

キ　商品コードをキーとする整列

設問2　あて先ファイル作成プログラムのモジュール構造を図3のとおりにし，モジュールの実行条件と処理を表に示すように設計した。表中の［　　］に入れる正しい答えを，解答群の中から選べ。

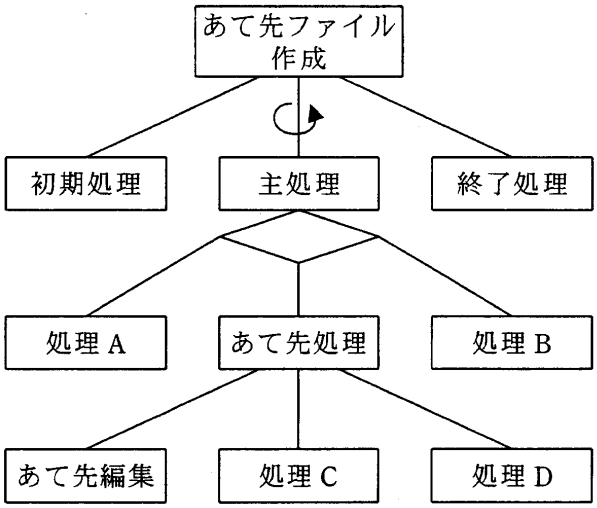


図3　モジュール構造図

表　モジュールの実行条件と処理

モジュール名	実行条件	処理
初期処理	−	必要な初期値設定を行う。中間ファイル及び応募者ファイルから最初の1レコードを読む。
主処理	［a］又は［b］まで	条件1～3によって，処理A，あて先処理，処理Bのどれかを実行する。
処理A	条件1	中間ファイルを読む。
あて先処理	条件2	あて先編集，処理C，処理Dを実行する。
処理B	条件3	応募者ファイルを読む。
あて先編集	−	顧客マスタファイルを読む。 あて先ファイルのレコードを編集する。 あて先ファイルを出力する。
処理C	−	中間ファイルを読む。
処理D	−	応募者ファイルを読む。
終了処理	−	ファイルを閉じるなどして，処理を終える。

注　条件1～3は，どれか一つだけが成立する。

解答群

ア　あて先ファイルが終わる　　イ　応募者ファイルが終わる

ウ　購入日付が変わる　　エ　購入日付が対象外となる

オ　顧客住所が変わる　　カ　顧客番号が変わる

キ　顧客マスタファイルが終わる　　ク　商品コードが変わる

ケ　中間ファイルが終わる

設問3　次のテストデータは，設問2の表中の条件1～3による分岐において，処理A，あて先処理，処理Bのすべてが実行されることを確かめるものである。このデータ中の［　　］に入れる正しい答えを，解答群の中から選べ。

中間ファイル

	購入日付	顧客番号	商品コード	購入数量	購入金額
1	20060315	c	XGD555	10	10500
2	20060330	d	YFQ886	2	3150

応募者ファイル

	顧客番号
1	400101
2	400103

注　購入日付の値は，どちらもキャンペーン対象期間内とする。

解答群

ア　400101　　イ　400102　　ウ　400103　　エ　400104

次の**問6**から**問9**までの 4 問については，この中から 1 問を選択し，答案用紙の選択欄の (選) をマークして解答してください。

なお，2 問以上選択した場合には，はじめの 1 問について採点します。

問6　次の C プログラムの説明及びプログラムを読んで，設問 1，2 に答えよ。

〔プログラムの説明〕

Helge von Koch が考案したコッホ曲線と呼ばれるフラクタル図形を描画する関数 KochCurve である。フラクタル図形は，図形の一部を拡大すると，再び，同様の図形が現れる自己相似性をもち，コッホ曲線はその代表的な図形である。

(1) 図 1 の①～③にコッホ曲線の生成手順を示す。

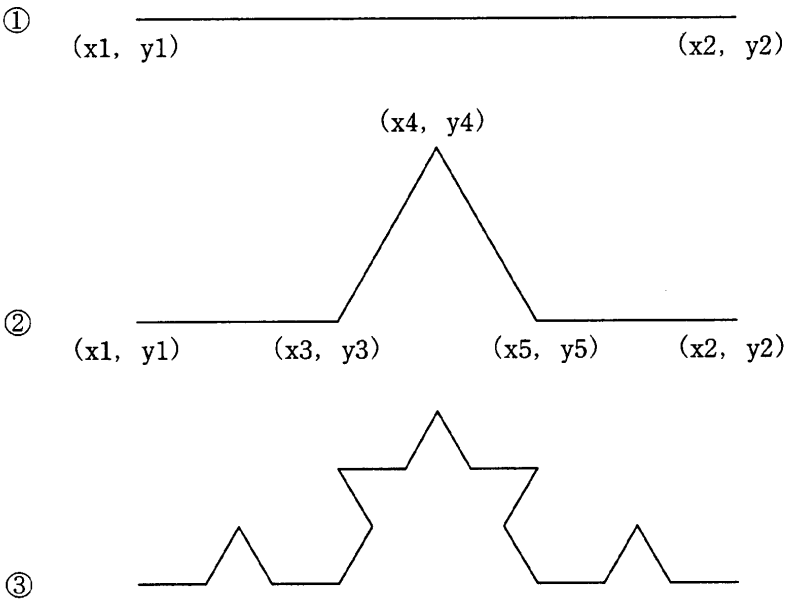


図 1　コッホ曲線の生成手順

① 始点の座標を (x1, y1)，終点を (x2, y2) とする 2 点を結ぶ線分を引く。

② 線分を 3 等分し，中央の線分を一辺とする正三角形を始点から終点に向かって左側に作り，その頂点座標を求める。始点 (x1, y1) から，正三角形の 3 頂点 (x3, y3)，(x4, y4)，(x5, y5)，そして終点 (x2, y2) を順番に結んだ四つの線分を引く。

③　得られた四つの線分それぞれを①の線分とみなし，②の操作を行う。

この操作を繰り返して得られるのがコッホ曲線である。

(2) 関数 DrawLine は，引数で指定された 2 点を結ぶ線分を描画する関数であり，既に用意されている。関数の仕様は，次のとおりである。

呼出し形式：

```
void DrawLine( int x1, int y1, int x2, int y2 );
```

引数：　x1 及び y1　線分の始点座標（x1, y1）

x2 及び y2　線分の終点座標（x2, y2）

返却値：　なし

(3) 関数 KochCurve は，引数で指定された 2 点（始点及び終点）の座標を基点にしてコッホ曲線を描画する関数であり，仕様は次のとおりである。

呼出し形式：

```
void KochCurve( int x1, int y1, int x2, int y2, int dim );
```

引数：　x1 及び y1　コッホ曲線の始点座標（x1, y1）

x2 及び y2　コッホ曲線の終点座標（x2, y2）

dim　(1) の説明での繰返し回数

0 のとき図 1 の①が描画される。

返却値：　なし

(4) このプログラムを実行すると，図 1 の②に示すような図形が描画される。また，行番号 25 を次のとおりに変更して実行すると，③に示すような図形が描画される。

```
25      KochCurve( 0, 180, 180, 180, 2 );
```

なお，座標軸は，原点（0, 0）から，右向きを x 軸の正方向，下向きを y 軸の正方向とする。

〔プログラム〕

(行番号)

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #define PI 3.141593

 4  void DrawLine( int, int, int, int );
 5  void KochCurve( int x1, int y1, int x2, int y2, int dim )
 6  {
 7      int x3, y3, x4, y4, x5, y5;

 8      if ( [ a ] )
 9          DrawLine( x1, y1, x2, y2 );
10      else {
11          x3 = ( 2 * x1 + x2 ) / 3;
12          y3 = ( 2 * y1 + y2 ) / 3;
13          x5 = ( x1 + 2 * x2 ) / 3;
14          y5 = ( y1 + 2 * y2 ) / 3;
15          x4 = x3 + (x5 - x3) * cos(PI/3) + (y5 - y3) * sin(PI/3);
16          y4 = y3 - (x5 - x3) * sin(PI/3) + (y5 - y3) * cos(PI/3);
17          KochCurve( x1, y1, x3, y3, [ b ] );
18          KochCurve( x3, y3, x4, y4, [ b ] );
19          KochCurve( x4, y4, x5, y5, [ b ] );
20          KochCurve( x5, y5, x2, y2, [ b ] );
21      }
22  }
23  void main()
24  {
25      KochCurve( 0, 180, 180, 180, 1 );
26  }
```

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　dim <= 0　　イ　dim <= 1　　ウ　dim == 1

エ　dim != 0　　オ　dim != 1

b に関する解答群

ア　dim--　　イ　dim - 1　　ウ　dim

エ　dim + 1　　オ　dim++

設問2　図2の図形を描画するためには，関数 main の行番号 25 をどのように変更すればよいか。正しい答えを，解答群の中から選べ。

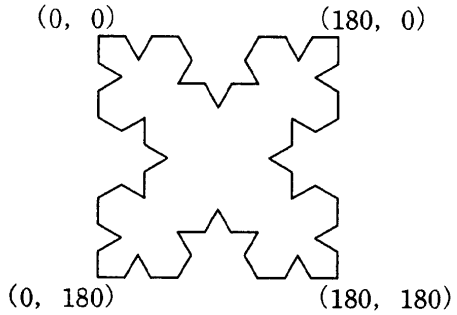


図2　描画したい図形

解答群

ア
```
KochCurve(   0,   0,   0, 180, 2 );
KochCurve(   0, 180, 180, 180, 2 );
KochCurve( 180, 180, 180,   0, 2 );
KochCurve( 180,   0,   0,   0, 2 );
```

イ
```
KochCurve(   0,   0, 180,   0, 2 );
KochCurve( 180,   0, 180, 180, 2 );
KochCurve( 180, 180,   0, 180, 2 );
KochCurve(   0,   0,   0, 180, 2 );
```

ウ
```
KochCurve(   0,   0, 180,   0, 2 );
KochCurve( 180,   0, 180, 180, 2 );
KochCurve( 180, 180,   0, 180, 2 );
KochCurve(   0, 180,   0,   0, 2 );
```

エ
```
KochCurve(   0,   0, 180,   0, 2 );
KochCurve( 180,   0, 180, 180, 2 );
KochCurve(   0,   0,   0, 180, 2 );
KochCurve(   0, 180, 180, 180, 2 );
```

オ
```
KochCurve( 180,   0, 180, 180, 2 );
KochCurve( 180, 180,   0, 180, 2 );
KochCurve(   0, 180,   0,   0, 2 );
KochCurve(   0,   0, 180,   0, 2 );
```

問7　次のCOBOLプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

駐輪場の使用状況を調べ，空き区画の番号を表示するプログラムである。駐輪場の各区画には自転車を1台駐輪することができ，それぞれの区画には区画番号が付けられている。区画は200台分あり，区画番号は1～200の連番となっている。

(1) 使用状況ファイルのレコード様式は，次のとおりである。

区画番号 3けた	その他の情報 100けた

① 区画番号の昇順に記録されている。

② 使用状況ファイルには，現在使用されている区画のレコードだけが記録されていて，同じ区画のレコードが重複して記録されることはない。

(2) 空き区画の情報を次のように表示する。

```
AKI KUKAKU
ZZZ
ZZZ - ZZZ
ZZZ - ZZZ
ZZZ
      ⋮
```

① 空き区画が連続していないときは，空き区画番号を表示する。

② 空き区画が連続する場合は，次の形式で表示する。

連続の始めの区画番号 - 連続の終わりの区画番号

③ 見出しは常に表示する。

④ 空き区画が一つもない場合は，次のメッセージを表示する。

```
AKI : 0
```

〔プログラム〕

```
DATA DIVISION.
FILE SECTION.
FD  SHIYO-F.
01  SHIYO-R.
    03  BANGO      PIC 9(3).
    03              PIC X(100).
WORKING-STORAGE SECTION.
01  END-SW         PIC X(3).
01  DISPLAY-SW     PIC X(2).
01  MIDASHI        PIC X(10) VALUE  "AKI KUKAKU".
01  MEISAI.
    03  M-HAJIME   PIC ZZZ.
    03  M-HYPHEN   PIC X(3).
    03  M-OWARI    PIC ZZZ.
01  W-SA           PIC 9(3).
01  W-BANGO        PIC 9(3).
PROCEDURE DIVISION.
SHORI.
    OPEN INPUT SHIYO-F.
    INITIALIZE W-BANGO END-SW DISPLAY-SW.
    DISPLAY MIDASHI.
    PERFORM UNTIL END-SW = "END"
       READ SHIYO-F AT END MOVE "END" TO END-SW
       NOT AT END
          COMPUTE W-SA = BANGO - W-BANGO
          IF W-SA > [      a      ] THEN
             MOVE SPACE TO MEISAI
             [      b      ]
             IF W-SA > [      c      ] THEN
                MOVE " - " TO M-HYPHEN
                [      d      ]
             END-IF
             DISPLAY MEISAI
             [      e      ]
          END-IF
          MOVE BANGO TO W-BANGO
       END-READ
    END-PERFORM.
    IF W-BANGO NOT = 200 THEN
       MOVE SPACE TO MEISAI
       [      b      ]
       IF W-BANGO < 199 THEN
          MOVE " - " TO M-HYPHEN
          MOVE 200 TO M-OWARI
       END-IF
       DISPLAY MEISAI
    ELSE IF DISPLAY-SW = SPACE THEN
       DISPLAY "AKI : 0"
    END-IF.
    CLOSE SHIYO-F.
    STOP RUN.
```

設問　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a，cに関する解答群

ア　0　　イ　1　　ウ　2

b，d，eに関する解答群

ア　COMPUTE M-HAJIME = BANGO - 1

イ　COMPUTE M-HAJIME = W-BANGO + 1

ウ　COMPUTE M-OWARI = BANGO - 1

エ　COMPUTE M-OWARI = W-BANGO + 1

オ　MOVE SPACE TO DISPLAY-SW

カ　MOVE SPACE TO MEISAI

キ　MOVE W-BANGO TO M-HAJIME

ク　MOVE W-BANGO TO M-OWARI

ケ　MOVE "ON" TO DISPLAY-SW

問8　次のJavaプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

PCのセキュリティに関する更新プログラム（以下，パッチという）の適用状況を管理するプログラムである。

クラス`PCChecker`は，PCに適用が要求されるパッチと各PCに適用済みのパッチを管理し，その適用状況の一覧を出力する。PCのパッチ情報として次のものが登録される。

① PCの識別子

② PCの種別（デスクトップPC又はノートPC）

③ 適用済みのパッチの識別子（複数の場合は複数個）

④ ノートPCの場合，暗号化ツールがインストールされているか否か

PCに問題がない（`secure`）と判断する基準は，次のとおりである。

デスクトップPC：	適用が要求されるパッチがすべて適用されている。
ノートPC：	適用が要求されるパッチがすべて適用されている。 さらに，暗号化ツールがインストールされている。

(1) クラス`PCChecker`は，PCのパッチ情報を管理するために，クラス`DesktopPC`とクラス`NotebookPC`のインスタンスを用いる。

(2) クラス`PC`は，デスクトップPCとノートPCに共通の処理を実現する。メソッド`installPatch`は，指定されたパッチを適用済みにする。メソッド`installed`は，指定されたパッチが適用済みのとき`true`，未適用のとき`false`を返す。メソッド`requiresEncryptionTool`は，暗号化ツールのインストールが必要なとき`true`，不必要なとき`false`を返す。

(3) クラス`DesktopPC`は，デスクトップPCを表し，デスクトップPC独自の処理を実現する。

(4) クラス`NotebookPC`は，ノートPCを表し，ノートPC独自の処理を実現する。

(5) クラス`PCChecker`のメソッド`main`は，このプログラムの使用例である。その実行結果を，次に示す。

```
pc001: requires [sp1-A01, sp1-A03]
pc002: requires encryption tool
pc003: secure
```

(6) クラス java.util.ArrayList は，オブジェクトをリストで管理する手段を提供する。次のメソッドを利用している。

public boolean add(Object obj)

オブジェクト obj をリストに追加し，true を返す。

public boolean contains(Object obj)

オブジェクト obj がリストに含まれる場合に true，含まれない場合に false を返す。

public boolean isEmpty()

リストが空の場合に true，空でない場合に false を返す。

〔プログラム1〕

```
import java.util.List;
import java.util.ArrayList;

public abstract class PC {
   private String id;
   private List installedPatches = new ArrayList();
   public PC(String id) { this.id = id; }
   public String getID() { return id; }
   public void installPatch(String patch) {
      installedPatches.add(patch);
   }
   public boolean installed(String patch) {
      return [      a      ];
   }
   public abstract boolean requiresEncryptionTool();
}
```

〔プログラム2〕

```
public class DesktopPC [      b      ] {
   public DesktopPC(String id) { [      c      ]; }
   public boolean requiresEncryptionTool() {
      return false;
   }
}
```

〔プログラム3〕

```
public class NotebookPC [      b      ] {
   private boolean hasEncryptionTool;
   public NotebookPC(String id, boolean hasEncryptionTool) {
      [      c      ];
      this.hasEncryptionTool = hasEncryptionTool;
   }
   public boolean requiresEncryptionTool() {
      return !hasEncryptionTool;
   }
}
```

〔プログラム4〕

```
import java.util.List;
import java.util.ArrayList;

public class PCChecker {
   public static void main(String[] args) {
      // PC のパッチ情報を登録する。
      PC[] pc = new PC[3];
      pc[0] = new DesktopPC("pc001");
      pc[1] = new NotebookPC("pc002", false);
      pc[2] = new NotebookPC("pc003", true);
      pc[0].installPatch("sp1-A02");
      pc[1].installPatch("sp1-A01");
      pc[1].installPatch("sp1-A02");
      pc[1].installPatch("sp1-A03");
      pc[2].installPatch("sp1-A01");
      pc[2].installPatch("sp1-A02");
      pc[2].installPatch("sp1-A03");
      // PC の状態を出力する。curPatches は，適用が要求されるパッチを示す。
      String[] curPatches = { "sp1-A01", "sp1-A02", "sp1-A03" };
      for (int i = 0; i < pc.length; i++) {
         System.out.print(pc[i].getID() + ":");
         List notApplied = new ArrayList();
         for (int j = 0; j < curPatches.length; j++)
            if ([      d      ])
               notApplied.add(curPatches[j]);
         if (!notApplied.isEmpty() ||
             pc[i].requiresEncryptionTool()) {
            if (!notApplied.isEmpty())
               System.out.println(" requires " + notApplied);
            if (pc[i].requiresEncryptionTool())
               System.out.println(" requires encryption tool");
         } else
            System.out.println(" secure");
      }
   }
}
```

設問　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　`installedPatches.isEmpty()`

イ　`!installedPatches.isEmpty()`

ウ　`installedPatches.contains(patch)`

エ　`!installedPatches.contains(patch)`

オ　`patch == null`

カ　`patch != null`

b に関する解答群

ア　`extends Object`　　イ　`extends PC`

ウ　`extends PCChecker`　　エ　`implements Object`

オ　`implements PC`　　カ　`implements PCChecker`

c に関する解答群

ア　`PC()`　　イ　`PC(id)`　　ウ　`super()`

エ　`super(id)`　　オ　`this.id = id`

d に関する解答群

ア　`curPatches[j] != null`

イ　`notApplied.isEmpty()`

ウ　`pc[i] != null`

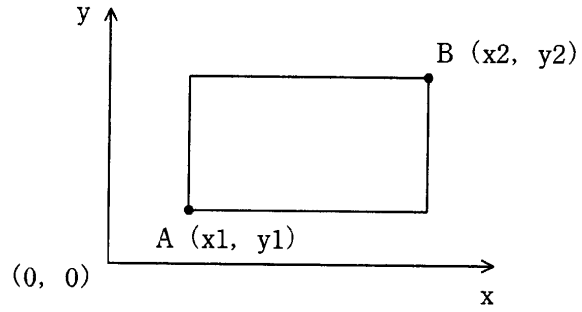
エ　`pc[i].installed(curPatches[j])`

オ　`!pc[i].installed(curPatches[j])`

問9　次のアセンブラプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラムの説明〕

(1) 副プログラムOBLONGは，各座標値が整数である2点A（x1，y1），B（x2，y2）を対角の頂点とし，各辺がx軸又はy軸に平行な長方形の面積を求めるプログラムである。



① 次のパラメタの先頭アドレスがGR1に格納されて，主プログラムから渡される。

(GR1)+0	x1
+1	y1
+2	x2
+3	y2

② x1≠x2，y1≠y2とする。

③ 求めた面積はGR0に格納して，主プログラムに返す。

④ 副プログラムOBLONGから戻るとき，汎用レジスタGR1～GR7の内容は元に戻す。

(2) 副プログラムABSは，OBLONGから渡されたGR2の内容をその絶対値に置き換えるプログラムである。

副プログラムABSの実行によって，汎用レジスタGR1，GR3～GR7の内容は変わらない。

(3) プログラム中の演算で，あふれは起きないものとする。

〔プログラム〕

```
(行番号)
   1  OBLONG   START
   2           RPUSH
   3           LD     GR2,0,GR1   ;┐
   4           SUBA   GR2,2,GR1   ; ├ GR3←|x1−x2|
   5           CALL   ABS         ; │
   6           LD     GR3,GR2     ;┘
   7           LD     GR2,1,GR1   ;┐
   8           SUBA   GR2,3,GR1   ; ├ GR2←|y1−y2|
   9           CALL   ABS         ;┘
  10           LD     GR0,=0      ; 面積の初期化
  11  LOOP     SRL    GR2,1
  12           [      a       ]
  13           JZE    FIN1
  14           [      b       ]
  15  ADD      ADDA   GR0,GR3
  16  SHIFT    SLA    GR3,1
  17           JUMP   LOOP
  18  FIN1     RPOP
  19           RET
  20  ;
  21  ABS      LD     GR0,GR2
  22           JMI    MINUS
  23           JUMP   FIN2
  24  MINUS    LD     GR2,=0
  25           [      c       ]
  26  FIN2     RET
  27           END
```

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

アセンブラ

a，b に関する解答群

ア	JMI	ADD	イ	JMI	LOOP	ウ	JMI	SHIFT
エ	JOV	ADD	オ	JOV	LOOP	カ	JOV	SHIFT
キ	JUMP	ADD	ク	JUMP	LOOP	ケ	JUMP	SHIFT

c に関する解答群

ア	ADDA	GR0,GR2	イ	ADDA	GR2,GR0
ウ	SUBA	GR0,GR2	エ	SUBA	GR2,GR0

設問2　2点AとBに次の座標を与えたとき，行番号15のADDA命令は何回実行されるか。正しい答えを，解答群の中から選べ。

Aの座標：（3，6）

Bの座標：（−5，17）

解答群

ア 1　　イ 2　　ウ 3　　エ 4　　オ 5

次の**問10**から**問13**までの4問については，この中から1問を選択し，答案用紙の選択欄の(選)をマークして解答してください。

なお，2問以上選択した場合には，はじめの1問について採点します。

問10　次のCプログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラムの説明〕

入力された英単語を検索し，訳語を表示するプログラムである。

(1) データは，木構造で登録されている。英単語を構成するそれぞれの文字を構造体 LETTER で表現する。構造体 LETTER のメンバ c には英小文字の文字符号が，メンバ follow にはその文字に続く文字を示す構造体 LETTER へのポインタが，メンバ other にはその文字に代わる文字を示す構造体 LETTER へのポインタが格納されている。メンバ trans には，p_root が指す根から順にたどってできる英単語に対する訳語の文字列へのポインタが格納されている。メンバ follow，other，trans には，該当するものがなければ空ポインタ定数（NULL）が格納されている。

例えば，五つの英単語“man”，“main”，“mail”，“mailer”，“name”が登録されている場合の構造は，図1のとおりとなる。

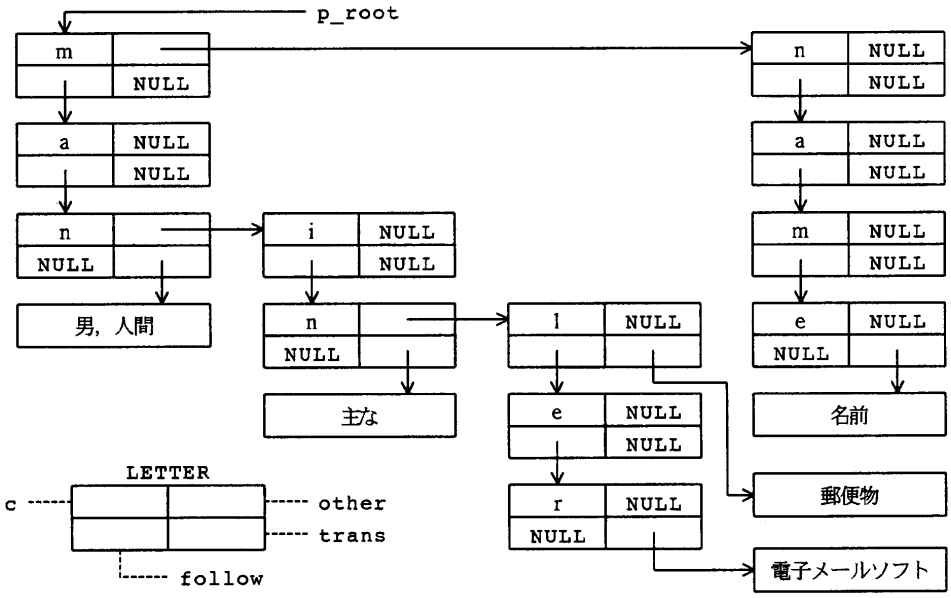


図1　英単語の登録例

(2) ユーザインタフェースは，図2に示すように，入力された英単語を表示するための検索語入力領域と，訳語を表示するための訳語表示領域からなる（以後，検索語入力領域に表示されている文字列を，入力文字列と呼ぶ)。

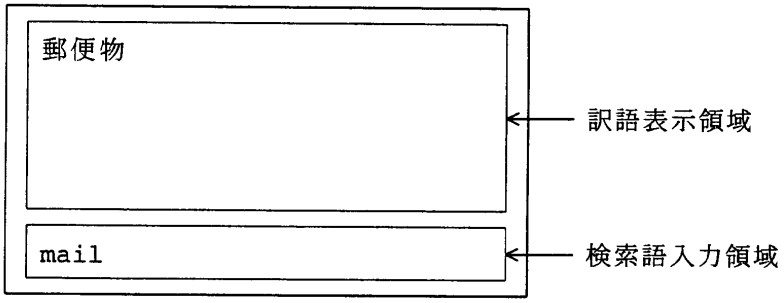


図2　ユーザインタフェースの例

(3) 入力文字列で始まる単語の候補がなくなった場合は，検索語入力の途中でも訳語表示領域上に“単語が見つかりません”を表示し，検索を終了する。

(4) ユーザインタフェースに関する次の関数が用意されているものとする。

```
void displayArea(char *s)
```

機能：訳語表示領域をクリアした後，文字列 s を表示する。

```
void clearField()
```

機能：検索語入力領域をクリアする。

```
void appendToField(char c)
```

機能：検索語入力領域に文字 c を追加表示する。

```
char inputChar()
```

機能：キーボードから1文字を読み込む。返却値は，入力された文字が英小文字の場合は文字符号，それ以外の場合は0となる。

〔プログラム〕

```
typedef struct letter {
    char c;                    /* 英単語を構成する文字 */
    struct letter *follow;     /* この文字に続く文字へのポインタ */
    struct letter *other;      /* この文字に代わる別の文字へのポインタ */
    char *trans;               /* 訳語の文字列へのポインタ */
} LETTER;

void displayArea(char *);
void clearField();
void appendToField(char);
char inputChar();

LETTER *p_root;

void searchWord(){
    LETTER *p = p_root;
    char c;
    clearField();              /* 検索語入力領域のクリア */
    displayArea("");           /* 訳語表示領域のクリア */
    while(c = inputChar()){    /* 1文字入力 */
        displayArea("");
        appendToField(c);      /* 検索語入力領域に入力文字を追加 */
        /* 終端又は文字のいずれかと一致するまで検索 */
        while((  a  ) && (p->c != c)){
            p = [ b ];
        }
        /* 文字と一致しなかった（入力文字列で始まる英単語がない）*/
        if(  c  ){
            displayArea("単語が見つかりません");
            return;
        }
α ──▶   /* 入力文字列と一致する英単語がある */
        if(p->trans != NULL){
            displayArea(p->trans);
        }
        p = [ d ];
    }
}
```

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a，cに関する解答群

ア　`p != NULL`　　イ　`p == NULL`

ウ　`p->follow != NULL`　　エ　`p->follow == NULL`

オ　`p->other != NULL`　　カ　`p->other == NULL`

キ　`p->trans != NULL`　　ク　`p->trans == NULL`

b，dに関する解答群

ア　`p->follow`　　イ　`p->follow->other`

ウ　`p->other`　　エ　`p->other->follow`

設問2　入力補完機能を追加することにした。例えば図1の登録例で，図3に示すように，初めに“m”を入力した場合は，それに一意に続く“a”を補完して検索語入力領域に追加表示する。また，初めに“n”を入力した場合は，“ame”を追加表示する。関数searchWordの修正は，次の表のとおりである。表中の［　　］に入れる正しい答えを，解答群の中から選べ。

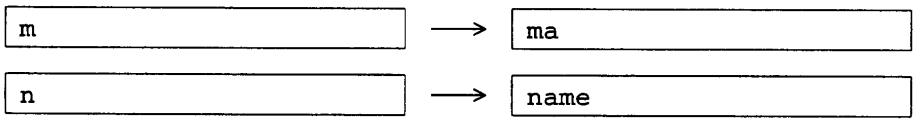


図3　補完例

処置	プログラムの変更内容

αの部分に追加	`while((p->trans == NULL) && (　　)){`
	`  p = p->follow;`
	`  appendToField(p->c);`
	`}`

解答群

ア　`p->follow != NULL`　　イ　`p->follow == NULL`

ウ　`p->follow->other != NULL`　　エ　`p->follow->other == NULL`

オ　`p->other != NULL`　　カ　`p->other == NULL`

問11　次の COBOL プログラムの説明及びプログラムを読んで，設問 1，2 に答えよ。

〔プログラムの説明〕

A ～ J の 10 駅を結ぶ路線をもつ鉄道において，利用者の入場駅，出場駅及び入場時刻が 1 日分記録された乗客ファイルを読み込み，各駅間の区間利用者数を集計して印字するプログラムである。

(1) 乗客ファイル（IN-FILE）は，次のレコード様式の順ファイルであり，乗客 1 人の 1 回の利用に対して一つのレコードが作成される。

入場駅 2 けた	出場駅 2 けた	入場時刻 4 けた

① 上り及び下り列車の利用者が順不同で格納されている。

② 入場時刻は，HHMM の形式で格納されている。ここで，HH は 24 時間表現の時を表し，MM は分を表す。

なお，各駅は午前 6 時に開場し，午前 0 時に閉鎖する。閉鎖中，乗客は駅及び列車内にとどまることはできない。

③ 入場駅及び出場駅は，次のとおり，それぞれの駅に対応する番号 01 ～ 10 として格納されている。入場駅と出場駅は異なるものとする。

A 駅は 01，B 駅は 02，C 駅は 03，D 駅は 04，…，J 駅は 10

例：6 時 55 分に B 駅から入場し，D 駅で出場した場合は，次のとおり記録される。

02	04	0655

(2) 印字様式は，次のとおりとする。

TIME		A-B	B-C	C-D	D-E	…	I-J
06:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
07:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
⋮							
23:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9

① TIME は，印字された時間帯の利用であることを表す。例えば，07 時 00 分～07 時 59 分における利用ならば，07:00 の行に集計される。

② A-B から I-J の各列は区間を表す。例えば，A-B は A 駅から B 駅までの区間を表す。

③ UP は上り（A 駅から J 駅の方向）を，DOWN は下り（J 駅から A 駅の方向）を表す。

④ 利用者数は，入場時刻を基にした経過時間及び乗車区間で集計する。

なお，各駅間の区間所要時間は 10 分とし，待ち時間や乗換えなどは考慮しない。例えば，(1) の例の“6 時 55 分に B 駅から入場し，D 駅で出場した場合”であれば，次の網掛け部分を 1 名が利用したとみなし，各要素にカウントする。人数は 999,999 以下とする。

TIME		A-B	B-C	C-D	D-E	…	I-J
06:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
07:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
⋮							
23:00	UP	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9
	DOWN	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	ZZZ,ZZ9	…	ZZZ,ZZ9

〔プログラム〕

(行番号)

```
 1  DATA DIVISION.
 2  FILE SECTION.
 3  FD  IN-FILE.
 4  01  IN-REC        PIC X(8).
 5  FD  PR-FILE.
 6  01  PR-REC        PIC X(100).
 7  WORKING-STORAGE SECTION.
 8  01  W-IN-REC.
 9      02 IN-STATION   PIC 9(2).
10      02 OUT-STATION  PIC 9(2).
11      02 IN-TIME.
12         03 IN-HH        PIC 9(2).
13         03 IN-MM        PIC 9(2).
14  01  COUNT-TABLE.
15      02 UP-TABLE    OCCURS 18.
16         03 UP-PASSENGER    OCCURS 9 PIC 9(6) VALUE ZERO.
```

```
17      02 DOWN-TABLE  OCCURS 18.
18         03 DOWN-PASSENGER  OCCURS 9 PIC 9(6) VALUE ZERO.
19  77  READ-SW      PIC  X(1) VALUE SPACE.
20      88 AT-END    VALUE "E".
21  77  WAY          PIC S9(1).
22      88 UP-WAY    VALUE  1.
23      88 DOWN-WAY  VALUE -1.
24  77  TIME-IDX     PIC S9(2).
25  77  TIME-CNT     PIC S9(2).
26  77  SECT-IDX     PIC S9(2).
27  01  W-PR-HEADER.
28      02               PIC X(15) VALUE "TIME".
29      02               PIC X(50) VALUE
30         "A-B       B-C       C-D       D-E       E-F".
31      02               PIC X(35) VALUE
32         "F-G       G-H       H-I       I-J".
33  01  W-PR-REC.
34      02 PR-TIME.
35         03 PR-HH      PIC 9(2).
36         03 PR-MM      PIC X(4).
37      02 PR-WAY        PIC X(4).
38      02               OCCURS 9.
39         03               PIC X(3) VALUE SPACE.
40         03 PR-PASSENGER  PIC ZZZ,ZZ9.
41  PROCEDURE DIVISION.
42  MAIN-PROCEDURE.
43      OPEN INPUT IN-FILE OUTPUT PR-FILE.
44      PERFORM TEST BEFORE UNTIL AT-END
45         READ IN-FILE AT END SET AT-END TO TRUE
46         NOT AT END MOVE IN-REC TO W-IN-REC
47                    PERFORM COUNT-PASSENGER
48         END-READ
49      END-PERFORM.
50      PERFORM PRINT-DATA.
51      CLOSE IN-FILE PR-FILE.
52      STOP RUN.
53  COUNT-PASSENGER.
54      IF [      a      ] THEN
55         SET UP-WAY   TO TRUE
56      ELSE
57         SET DOWN-WAY TO TRUE
58      END-IF.
59      COMPUTE TIME-IDX = IN-HH - 5.
60      MOVE IN-MM TO TIME-CNT.
61      PERFORM VARYING SECT-IDX FROM IN-STATION BY WAY
62              UNTIL SECT-IDX = OUT-STATION
63         IF UP-WAY THEN
64            ADD 1 TO UP-PASSENGER(TIME-IDX, SECT-IDX)
65         ELSE
66            ADD 1 TO DOWN-PASSENGER([      b      ])
67         END-IF
68         COMPUTE TIME-CNT = TIME-CNT + 10
```

```
69          IF [      c      ] THEN
70             COMPUTE TIME-IDX = TIME-IDX + 1
71             COMPUTE TIME-CNT = TIME-CNT - 60
72          END-IF
73       END-PERFORM.
74   PRINT-DATA.
75       WRITE PR-REC FROM W-PR-HEADER.
76       PERFORM VARYING TIME-IDX FROM 1 BY 1
77               UNTIL TIME-IDX > 18
78          COMPUTE PR-HH = TIME-IDX + 5
79          MOVE ":00" TO PR-MM
80          MOVE "UP"  TO PR-WAY
81          PERFORM VARYING SECT-IDX FROM 1 BY 1
82                  UNTIL SECT-IDX > 9
83             MOVE UP-PASSENGER(TIME-IDX, SECT-IDX)
84               TO PR-PASSENGER([      d      ])
85          END-PERFORM
86          WRITE PR-REC FROM W-PR-REC
87          MOVE SPACE  TO PR-TIME
88          MOVE "DOWN" TO PR-WAY
89          PERFORM VARYING SECT-IDX FROM 1 BY 1
90                  UNTIL SECT-IDX > 9
91             MOVE DOWN-PASSENGER(TIME-IDX, SECT-IDX)
92               TO PR-PASSENGER([      d      ])
93          END-PERFORM
94          WRITE PR-REC FROM W-PR-REC
95       END-PERFORM.
```

COBOL

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a，cに関する解答群

ア	IN-STATION < 5	イ	IN-STATION < OUT-STATION
ウ	IN-STATION > 5	エ	IN-STATION > OUT-STATION
オ	TIME-CNT = 60	カ	TIME-CNT >= 60
キ	TIME-IDX = 0	ク	TIME-IDX NOT = 0

b，dに関する解答群

ア	SECT-IDX	イ	SECT-IDX, TIME-IDX
ウ	TIME-IDX - 1, SECT-IDX	エ	TIME-IDX + 1, SECT-IDX
オ	TIME-IDX	カ	TIME-IDX, SECT-IDX - 1
キ	TIME-IDX, SECT-IDX + 1	ク	TIME-IDX, SECT-IDX

設問2　区間利用者数をより正確に集計するため，各駅間の所要時間を分単位で保持するテーブルを用意し，そのテーブルから経過時間を求めるよう，プログラムを変更したい。次の表中の［　　］に入れる正しい答えを，解答群の中から選べ。

処置	プログラムの変更内容
行番号 40 と 41 の間に追加	01 TIME-TABLE. 02 TIME-DATA PIC X(18) VALUE "080611071009081013". 02 SECT-MM REDEFINES TIME-DATA PIC 9(2) OCCURS 9.
行番号 64 と 65 の間に追加	COMPUTE TIME-CNT = ［ e ］
行番号 66 と 67 の間に追加	COMPUTE TIME-CNT = ［ f ］
行番号 68 を削除	

COBOL

解答群

ア　60 - SECT-MM(SECT-IDX)

イ　60 - SECT-MM(TIME-IDX)

ウ　TIME-CNT + SECT-MM(SECT-IDX)

エ　TIME-CNT + SECT-MM(SECT-IDX + 1)

オ　TIME-CNT + SECT-MM(SECT-IDX - 1)

カ　TIME-CNT + SECT-MM(TIME-IDX)

キ　TIME-CNT + SECT-MM(TIME-IDX + 1)

ク　TIME-CNT + SECT-MM(TIME-IDX - 1)

問12 次のJavaプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

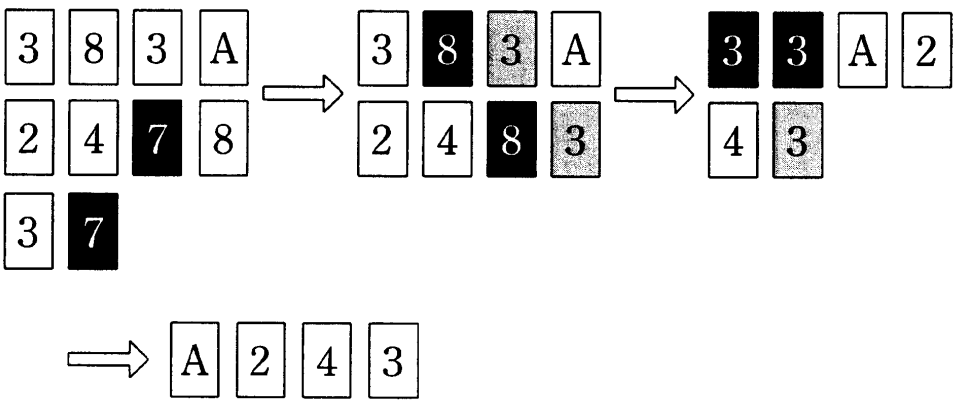
トランプを用いた一人遊びのゲームを実行するプログラムである。ゲームのルールは，次のとおりである。

(1) トランプは，スペード，ハート，ダイヤ及びクラブの4種類のスートがあり，各スートはA（エース），2～10，J（ジャック），Q（クイーン）及びK（キング）の1～13の位（ランク）を表す13枚のカードからなる。このゲームでは合計52枚のカードを使用し，スートは区別しない。

(2) 52枚のカードを，1枚ずつ表を向けて横に並べていく。4枚並べたら，その下の列に移り，再び左端から横に並べる。カードを並べていくところを場と呼び，横の並びを列と呼ぶ。

(3) 図に示すとおり，(2)で1枚並べるごとに，縦，横，斜めに隣り合ったカードが同じ位であるかどうかを調べ，同じ位のカードの組み（ペア）ができた場合は，そのペア（図中の 7 ）を取り除き，空いたスペースを詰める。列の最後のカードと次の列の最初のカードは隣り合ったカードとはみなさない。ペアが複数できた場合（ 8 と 3 の各2枚）は，最も長く場にあるカードを含むペア（ 8 ）を取り除く。最も長く場にあるカードを含むペアが複数ある場合は，最も長く場にあるカードとペアを作るカードのうちで，最も長く場にあるカードとのペア（ 3 ）を取り除く。

(4) (3)の操作を隣り合うカードの位が全部異なるようになるまで繰り返す。



Java

(5) (2) ～ (4) の操作を手持ちのカードがなくなるまで繰り返す。

(6) 手持ちのカードがなくなったときに，すべてのカードが場から取り除かれると上がりである。

クラス Card は，トランプのカードを表す。クラスの初期化のとき，A（エース）～ K（キング）の位に相当する Card のインスタンスを 4 種類のスート分生成して Card の配列 cards に格納する。Card のインスタンスは不変であり，1 枚のカードに必ず同一のインスタンスが対応する。例えば，ハートのエースを表す Card のインスタンスは一つしか存在しない。クラスメソッド newDeck は，cards をランダムな順番に並べ替えた Card の配列をトランプの一山として返す。

クラス Game は，ゲームを実行するプログラムである。List のインスタンス list がトランプを並べていく場を表し，トランプの山を表す deck から 1 枚ずつ list に追加し，その都度メソッド checkAndRemove を呼び出して同じ位の隣り合うカードのペアを取り除く。

クラス java.util.Random は，乱数を生成するためのクラスである。メソッド nextInt(int n) は，範囲 0 ～ n−1 の乱数を int 型で返す。

インタフェース java.util.List は，リスト構造を表し，各要素はインデックスで指定される。リストの最初の要素は，インデックスの値 0 で指定される。メソッド add(Object obj) は，リストの最後にオブジェクト obj を追加する。メソッド get(int index) は，index で指定された要素のオブジェクトを返す。メソッド remove(int index) は，index で指定された要素のオブジェクトを削除し，index + 1 以降にオブジェクトがあれば，それらをシフトして空きを詰める。メソッド size() は，リストにあるオブジェクトの個数を int 型で返す。

クラス java.util.ArrayList は，配列を用いてインタフェース List を実装する。

〔プログラム1〕

```
import java.util.Random;

public class Card {
   public static final int SPADES = 0;
   public static final int HEARTS = 1;
   public static final int DIAMONDS = 2;
   public static final int CLUBS = 3;

   private static final Random rand = new Random();
   private static final Card[] cards = new Card[13 * 4];
   private final int suit;
   private final int rank;

   [          a          ]
      for (int i = SPADES; i <= CLUBS; i++) {
         for (int j = 1; j <= 13; j++) {
            cards[          b          ] = new Card(i, j);
         }
      }
   }

   private Card(int suit, int rank) {
      this.suit = suit;
      this.rank = rank;
   }

   public int getSuit() { return suit; }
   public int getRank() { return rank; }

   public static Card[] newDeck() {
      Card[] deck = new Card[cards.length];
      System.arraycopy(cards, 0, deck, 0, cards.length);
      for (int i = cards.length - 1; i > 0; i--) {
         int index = rand.nextInt(i + 1);
         Card tmp = deck[i];
         deck[i] = deck[index];
         deck[index] = tmp;
      }
      return deck;
   }
}
```

〔プログラム2〕

```
import java.util.ArrayList;
import java.util.List;
```

```
public class Game {
   // 現在注目しているカードからの相対インデックス
   private static final int[][] indexDiff = {
      { 1, 4, 5 },
      { 1, 3, 4, 5 },
      { 1, 3, 4, 5 },
      { 3, 4 }
   };

   private static void checkAndRemove(List list) {
      // list の最初からその隣り合うカードを調べる。
      // currentIndex は現在注目しているカードのインデックス値
      for (int currentIndex = 0; currentIndex < list.size();
           currentIndex++) {
         // currentRank は現在注目しているカードの位
         int currentRank =
                   ((Card) list.get(currentIndex)).getRank();
         // 現在注目しているカードと隣り合うカードへの
         // インデックスを求める。
         int[] diffList = indexDiff[currentIndex % 4];
         for (int i = 0; i < diffList.length; i++) {
            // adjacentIndex は隣り合うカードへのインデックス値
            int adjacentIndex = [          c          ];
            if ([          d          ]
                && ((Card) list.get(adjacentIndex)).getRank()
                    == currentRank) {
               list.remove(adjacentIndex);
               list.remove([          e          ]);
               checkAndRemove(list);
               return;
            }
         }
      }
   }

   public static void main(String[] args) {
      Card[] deck = Card.newDeck();
      List list = new ArrayList();
      for (int i = 0; i < deck.length; i++) {
         list.add(deck[i]);
         checkAndRemove(list);
      }
      if (list.size() == 0) {
         System.out.println("上がり！");
      } else {
         System.out.println("残り" + list.size() + "枚");
      }
   }
}
```

設問　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　`private Card() {`　　イ　`private static void init() {`

ウ　`private void init() {`　　エ　`static {`

オ　`synchronized {`　　カ　`{`

b に関する解答群

ア　`i * 13 + j`　　イ　`i * 13 + j - 1`

ウ　`i * 4 + j - 1`　　エ　`j * 13 + i`

オ　`j * 13 + i - 1`　　カ　`j * 4 + i - 1`

c に関する解答群

ア　`currentIndex * diffList[i]`　　イ　`currentIndex + diffList[i]`

ウ　`currentIndex - diffList[i]`　　エ　`i * diffList[currentIndex]`

オ　`i + diffList[currentIndex]`　　カ　`i - diffList[currentIndex]`

d に関する解答群

ア　`adjacentIndex != list.size()`

イ　`adjacentIndex < list.size()`

ウ　`adjacentIndex <= list.size()`

エ　`adjacentIndex == list.size()`

オ　`adjacentIndex > list.size()`

カ　`adjacentIndex >= list.size()`

e に関する解答群

ア　`0`　　イ　`adjacentIndex`

ウ　`adjacentIndex - 1`　　エ　`currentIndex`

オ　`currentIndex - 1`　　カ　`list.size() - 1`

Java

問13　次のアセンブラプログラムの説明及びプログラムを読んで，設問1～3に答えよ。

〔プログラムの説明〕

32ビット符号なし2進整数と15ビット符号なし2進整数の乗算を行う副プログラムMULT32と，32ビット符号なし2進整数同士の加算を行う副プログラムADD32である。

(1) MULT32は，32ビットの被乗数が格納されている領域の先頭番地をGR2に，15ビットの乗数をGR1に設定して呼び出される。MULT32は，32ビットの乗算結果を被乗数の格納領域に上書きして呼出し元に戻る。乗算結果のうち，格納領域からはみ出した部分は無視する。MULT32は，その処理の過程でADD32を利用する。

(2) ADD32は，被加数の格納されている領域の先頭番地をGR2に，加数の上位語と下位語をそれぞれGR1とGR0に設定して呼び出される。ADD32は，32ビットの加算結果を被加数の格納領域に上書きして呼出し元に戻る。加算結果のうち，格納領域からはみ出した部分は無視する。

(3) 被乗数と被加数の格納形式を，次に示す。

上位語	下位語
(GR2)+0	(GR2)+1

(4) 副プログラムから戻るとき，汎用レジスタGR1～GR7の内容は元に戻す。

アセンブラ

〔プログラム〕

(行番号)

```
1  MULT32  START             ; シフトによる乗算
2          RPUSH
3          LD    GR5,0,GR2   ;
4          ST    GR5,H       ; 被乗数の退避
5          LD    GR5,1,GR2   ;
6          ST    GR5,L       ;
7          LD    GR5,=0      ;
8          ST    GR5,0,GR2   ; 乗算結果格納領域の初期化
9          ST    GR5,1,GR2   ;
```

```
10              LAD     GR3,15      ; 被乗数シフト用カウンタ初期化
11              LAD     GR4,1       ; GR4 ← 16−GR3
12              LD      GR6,GR1     ; 乗数を GR6 に設定，GR1 は作業用に解放
13  LP          LAD     GR3,-1,GR3
14              LAD     GR4,1,GR4   ; GR3＋GR4＝16 の関係を維持
15              SLL     GR6,1
16              JZE     FIN
17              JPL     LP          ; 最上位ビットが 0 なら何もしない
18              LD      GR1,H       ; 被乗数の取出し
19              LD      GR0,L
20              LD      GR5,GR0     ;
21              [   a   ]           ;
22              SLL     GR1,0,GR3   ; } GR3 だけ被乗数（32 ビット）を左シフト
23              SLL     GR0,0,GR3   ;
24              OR      GR1,GR5     ;
25              CALL    ADD32       ; シフトされた被乗数を中間結果に加算
26              JUMP    LP
27  FIN         RPOP
28              RET
29  H           DS      1           ; 被乗数上位語の退避場所
30  L           DS      1           ;   〃   下位語    〃
31  ;
32  ADD32       RPUSH               ; Z ← X＋Y
33                                  ; X の上位語を XH，下位語を XL などと表記
34              LD      GR4,GR0     ; YL を GR4 に設定
35              ADDL    GR0,1,GR2   ; ZL ← XL＋YL，ZL を GR0 に設定
36              XOR     GR4,1,GR2   ; けた上がりの判定
37              JMI     DIFF        ; ① XL と YL の最上位ビットが，
38              XOR     GR4,1,GR2   ;   ともに 1 のときけた上がり
39              JUMP    NEXT        ; ② XL と YL の最上位ビットが異なり，
40  DIFF        XOR     GR4,GR0     ;   ZL の最上位ビットが 0 のときけた上がり
41  NEXT        SRL     GR4,15      ; けた上がりを GR4 の最下位ビットへ移動
42              [   b   ]
43              ADDL    GR1,0,GR2
44              ST      GR1,0,GR2
45              ST      GR0,1,GR2
46              RPOP
47              RET
48              END
```

設問 1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

a に関する解答群

ア　SLL　GR5,0,GR3	イ　SLL　GR5,0,GR4	ウ　SRA　GR5,0,GR3
エ　SRA　GR5,0,GR4	オ　SRL　GR5,0,GR3	カ　SRL　GR5,0,GR4

b に関する解答群

ア	ADDL GR0,GR4	イ	ADDL GR1,GR4	ウ	OR GR0,GR4
エ	OR GR1,GR4	オ	SUBL GR0,GR4	カ	SUBL GR1,GR4

設問2 次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

MULT32 の受け取る乗数を 16 ビット符号なし 2 進整数とするため，行番号 10，11 を次のとおりに変更する。

```
10          LAD   GR3,16      ; 被乗数シフト用カウンタ初期化
11          LAD   GR4,0       ; GR4 ← 16−GR3
```

さらに，行番号 16，17 を次の命令群で置き換える。

```
            [          ]
            JUMP  LP
ONBIT       NOP
```

解答群

ア	JOV LP JPL ONBIT	イ	JOV ONBIT JZE FIN	ウ	JPL FIN JOV ONBIT
エ	JPL ONBIT JZE FIN	オ	JZE FIN JOV ONBIT	カ	JZE FIN JPL ONBIT

アセンブラ

設問3　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

MULT32 を使用して，再帰的に階乗計算を行う副プログラム FACT を作成した。

(1) 正の整数 n の階乗 F(n) は，次式で求められる。

$F(n) = n \times F(n-1)$　（ただし，$F(0) = 1$）

例：$F(3) = 3 \times F(2) = 3 \times 2 \times F(1) = 3 \times 2 \times 1 \times F(0) = 3 \times 2 \times 1 \times 1 = 6$

(2) FACT は，n を GR1 に，計算結果を格納する領域の先頭番地を GR2 に設定して呼び出される。n は，その階乗 F(n) が 32 ビット符号なし 2 進整数の範囲に収まるように与えられる（$1 \leqq n \leqq 12$）。計算結果格納領域の形式は，MULT32 の被乗数の格納形式と同じとする。

(3) 副プログラムから戻るとき，汎用レジスタ GR1 ～ GR7 の内容は元に戻す。

（行番号）

```
 1  FACT   START
 2         RPUSH
 3         CALL RMAIN          ; 再帰処理の本体を呼ぶ
 4         RPOP
 5         RET                 ; 主プログラムに戻る
 6  RMAIN  LD   GR1,GR1        ; N=0？
 7         JNZ  CONT           ; N≠0 なら CONT へ
 8         ST   GR1,0,GR2      ; N=0 の場合
 9         LD   GR1,=1         ;   F(0) の値 (=1) を
10         ST   GR1,1,GR2      ;   計算結果格納領域に設定し
11         RET                 ;   行番号 15 に戻る
12  CONT   PUSH 0,GR1          ; N の退避（N は n → n−1 →…→ 1 の順に変化）
13         LAD  GR1,-1,GR1     ; GR1 ← N−1
14         CALL RMAIN          ; F(N−1) を計算し計算結果格納領域に設定
15         POP  GR1            ; N の復元（N は 1 → 2 →…→ n の順に変化）
16         CALL MULT32         ; F(N−1)×N を計算し計算結果格納領域に設定
17         RET                 ; 再帰処理の終了時は行番号 4 に，
18         END                 ;   それ以外のときは行番号 15 に戻る
```

F(3) を計算するとき，行番号 6 のラベル RMAIN の命令には［　c　］回制御が移る。また，行番号 11 の RET 命令は［　d　］回実行される。

解答群

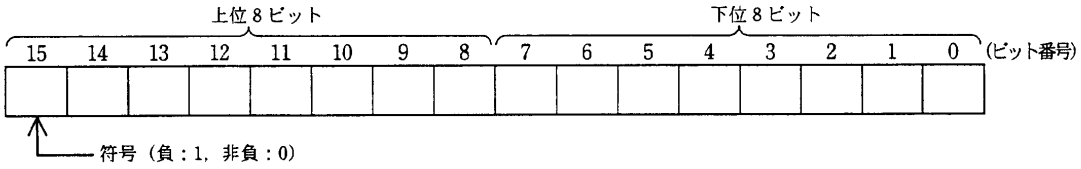
ア　1	イ　2	ウ　3
エ　4	オ　5	カ　6

■アセンブラ言語の仕様

1. システム COMETⅡの仕様

1.1 ハードウェアの仕様

(1) 1語は16ビットで，そのビット構成は，次のとおりである。



(2) 主記憶の容量は65536語で，そのアドレスは0～65535番地である。

(3) 数値は，16ビットの2進数で表現する。負数は，2の補数で表現する。

(4) 制御方式は逐次制御で，命令語は1語長又は2語長である。

(5) レジスタとして，GR（16ビット），SP（16ビット），PR（16ビット），FR（3ビット）の4種類がある。

GR（汎用レジスタ，General Register）は，GR0～GR7 の 8 個があり，算術，論理，比較，シフトなどの演算に用いる。このうち，GR1～GR7 のレジスタは，指標レジスタ(index register）としてアドレスの修飾にも用いる。

SP（スタックポインタ，Stack Pointer）は，スタックの最上段のアドレスを保持している。

PR（プログラムレジスタ，Program Register）は，次に実行すべき命令語の先頭アドレスを保持している。

FR（フラグレジスタ，Flag Register）は，OF（Overflow Flag），SF（Sign Flag），ZF (Zero Flag）と呼ぶ3個のビットからなり，演算命令などの実行によって次の値が設定される。これらの値は，条件付き分岐命令で参照される。

OF	算術演算命令の場合は，演算結果が－32768～32767 に収まらなくなったとき 1 になり，それ以外のとき 0 になる。論理演算命令の場合は，演算結果が 0～65535 に収まらなくなったとき 1 になり，それ以外のとき 0 になる。
SF	演算結果の符号が負（ビット番号 15 が 1）のとき 1，それ以外のとき 0 になる。
ZF	演算結果が零（全部のビットが 0）のとき 1，それ以外のとき 0 になる。

(6) 論理加算又は論理減算は，被演算データを符号のない数値とみなして，加算又は減算する。

1.2 命令

命令の形式及びその機能を示す。ここで，一つの命令コードに対し 2 種類のオペランドがある場合，上段はレジスタ間の命令，下段はレジスタと主記憶間の命令を表す。

命　令	書き方		命令の説明	FRの設定
	命令コード	オペランド		

(1) ロード，ストア，ロードアドレス命令

ロード LoaD	LD	r1,r2	r1 ← (r2)	○*1
		r,adr [,x]	r ← (実効アドレス)	
ストア STore	ST	r,adr [,x]	実効アドレス ← (r)	－
ロードアドレス Load ADdress	LAD	r,adr [,x]	r ← 実効アドレス	

(2) 算術，論理演算命令

算術加算 ADD Arithmetic	ADDA	r1,r2	r1 ← (r1) + (r2)	○
		r,adr [,x]	r ← (r) + (実効アドレス)	
論理加算 ADD Logical	ADDL	r1,r2	r1 ← (r1) $+_L$ (r2)	
		r,adr [,x]	r ← (r) $+_L$ (実効アドレス)	
算術減算 SUBtract Arithmetic	SUBA	r1,r2	r1 ← (r1) − (r2)	
		r,adr [,x]	r ← (r) − (実効アドレス)	
論理減算 SUBtract Logical	SUBL	r1,r2	r1 ← (r1) $-_L$ (r2)	
		r,adr [,x]	r ← (r) $-_L$ (実効アドレス)	
論理積 AND	AND	r1,r2	r1 ← (r1) AND (r2)	○*1
		r,adr [,x]	r ← (r) AND (実効アドレス)	
論理和 OR	OR	r1,r2	r1 ← (r1) OR (r2)	
		r,adr [,x]	r ← (r) OR (実効アドレス)	
排他的論理和 eXclusive OR	XOR	r1,r2	r1 ← (r1) XOR (r2)	
		r,adr [,x]	r ← (r) XOR (実効アドレス)	

(3) 比較演算命令

算術比較 ComPare Arithmetic	CPA	r1,r2 r,adr [,x]	(r1) と (r2)，又は (r) と (実効アドレス) の算術比較又は論理比較を行い，比較結果によって，FR に次の値を設定する。	○*1
論理比較 ComPare Logical	CPL	r1,r2 r,adr [,x]	

比較結果	FR の値 SF	FR の値 ZF
(r1) ＞ (r2) (r) ＞ (実効アドレス)	0	0
(r1) ＝ (r2) (r) ＝ (実効アドレス)	0	1
(r1) ＜ (r2) (r) ＜ (実効アドレス)	1	0

(4) シフト演算命令

算術左シフト Shift Left Arithmetic	SLA	r,adr [,x]	符号を除き (r) を実効アドレスで指定したビット数だけ左又は右にシフトする。 シフトの結果，空いたビット位置には，左シフトのときは 0，右シフトのときは符号と同じものが入る。	○*2
算術右シフト Shift Right Arithmetic	SRA	r,adr [,x]		
論理左シフト Shift Left Logical	SLL	r,adr [,x]	符号を含み (r) を実効アドレスで指定したビット数だけ左又は右にシフトする。 シフトの結果，空いたビット位置には 0 が入る。	
論理右シフト Shift Right Logical	SRL	r,adr [,x]		

(5) 分岐命令

正分岐 Jump on PLus	JPL	adr [,x]	FR の値によって，実効アドレスに分岐する。分岐しないときは，次の命令に進む。	—
負分岐 Jump on MInus	JMI	adr [,x]		
非零分岐 Jump on Non Zero	JNZ	adr [,x]		
零分岐 Jump on ZEro	JZE	adr [,x]		
オーバフロー分岐 Jump on OVerflow	JOV	adr [,x]		
無条件分岐 unconditional JUMP	JUMP	adr [,x]	無条件に実効アドレスに分岐する。	

命令	分岐するときの FR の値 OF	分岐するときの FR の値 SF	分岐するときの FR の値 ZF
JPL		0	0
JMI		1	
JNZ			0
JZE			1
JOV	1		

(6) スタック操作命令

プッシュ PUSH	PUSH adr [,x]	SP ← (SP) $-_L$ 1, (SP) ← 実効アドレス	—
ポップ POP	POP r	r ← ((SP)), SP ← (SP) $+_L$ 1	

(7) コール，リターン命令

コール CALL subroutine	CALL adr [,x]	SP ← (SP) $-_L$ 1, (SP) ← (PR), PR ← 実効アドレス	—
リターン RETurn from subroutine	RET	PR ← ((SP)), SP ← (SP) $+_L$ 1	

(8) その他

スーパバイザコール SuperVisor Call	SVC adr [,x]	実効アドレスを引数として割出しを行う。実行後の GR と FR は不定となる。	—
ノーオペレーション No OPeration	NOP	何もしない。	

(注) r, r1, r2　いずれも GR を示す。指定できる GR は GR0 ～ GR7
adr　アドレスを示す。指定できる値の範囲は 0 ～ 65535
x　指標レジスタとして用いる GR を示す。指定できる GR は GR1 ～ GR7
[　]　[　] 内の指定は省略できることを示す。
(　)　(　) 内のレジスタ又はアドレスに格納されている内容を示す。
実効アドレス　adr と x の内容との論理加算値又はその値が示す番地
←　演算結果を，左辺のレジスタ又はアドレスに格納することを示す。
$+_L$, $-_L$　論理加算，論理減算を示す。
FR の設定　○　：設定されることを示す。
○*1：設定されることを示す。ただし，OF には 0 が設定される。
○*2：設定されることを示す。ただし，OF にはレジスタから最後に送り出されたビットの値が設定される。
－　：実行前の値が保持されることを示す。

1.3 文字の符号表

(1) JIS X 0201 ラテン文字・片仮名用 8 ビット符号で規定する文字の符号表を使用する。
(2) 右に符号表の一部を示す。1 文字は 8 ビットからなり，上位 4 ビットを列で，下位 4 ビットを行で示す。例えば，間隔，4，H，¥ のビット構成は，16 進表示で，それぞれ 20，34，48，5C である。16 進表示で，ビット構成が 21 ～ 7E（及び表では省略している A1 ～ DF）に対応する文字を図形文字という。図形文字は，表示（印刷）装置で，文字として表示（印字）できる。
(3) この表にない文字とそのビット構成が必要な場合は，問題中で与える。

行＼列	02	03	04	05	06	07
0	間隔	0	@	P	`	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
10	*	:	J	Z	j	z
11	+	;	K	[	k	{
12	,	<	L	¥	l	|
13	-	=	M	]	m	}
14	.	>	N	^	n	~
15	/	?	O	_	o	

2. アセンブラ言語 CASLⅡの仕様

2.1 言語の仕様

(1) CASLⅡは，COMETⅡのためのアセンブラ言語である。
(2) プログラムは，命令行及び注釈行からなる。
(3) 1命令は1命令行で記述し，次の行へ継続できない。
(4) 命令行及び注釈行は，次に示す記述の形式で，行の1文字目から記述する。

行の種類		記述の形式
命令行	オペランドあり	[ラベル] {空白} {命令コード} {空白} {オペランド} [{空白} [コメント]]
	オペランドなし	[ラベル] {空白} {命令コード} [{空白} [{;} [コメント]]]
注釈行		[空白] {;} [コメント]

(注) [ ] [ ] 内の指定が省略できることを示す。
{ } { } 内の指定が必須であることを示す。
ラベル その命令の（先頭の語の）アドレスを他の命令やプログラムから参照するための名前である。長さは 1～8 文字で，先頭の文字は英大文字でなければならない。以降の文字は，英大文字又は数字のいずれでもよい。なお，予約語である GR0～GR7 は，使用できない。
空白 1文字以上の間隔文字の列である。
命令コード 命令ごとに記述の形式が定義されている。
オペランド 命令ごとに記述の形式が定義されている。
コメント 覚え書きなどの任意の情報であり，処理系で許す任意の文字を書くことができる。

2.2 命令の種類

命令は，4種類のアセンブラ命令（START，END，DS，DC），4種類のマクロ命令（IN，OUT，RPUSH，RPOP）及び機械語命令（COMETⅡの命令）からなる。その仕様を次に示す。

命令の種類	ラベル	命令コード	オペランド	機能
アセンブラ命令	ラベル	START	[実行開始番地]	プログラムの先頭を定義 プログラムの実行開始番地を定義 他のプログラムで参照する入口名を定義
		END		プログラムの終わりを明示
	[ラベル]	DS	語数	領域を確保
	[ラベル]	DC	定数 [,定数] …	定数を定義
マクロ命令	[ラベル]	IN	入力領域,入力文字長領域	入力装置から文字データを入力
	[ラベル]	OUT	出力領域,出力文字長領域	出力装置へ文字データを出力
	[ラベル]	RPUSH		GR の内容をスタックに格納
	[ラベル]	RPOP		スタックの内容を GR に格納
機械語命令	[ラベル]	（「1.2 命令」を参照）		

2.3 アセンブラ命令

アセンブラ命令は，アセンブラの制御などを行う。

(1) START [実行開始番地]

START 命令は，プログラムの先頭を定義する。

実行開始番地は，そのプログラム内で定義されたラベルで指定する。指定がある場合はその番地から，省略した場合は START 命令の次の命令から，実行を開始する。

また，この命令につけられたラベルは，他のプログラムから入口名として参照できる。

(2) | END | |

END 命令は，プログラムの終わりを定義する。

(3) | DS | 語数 |

DS 命令は，指定した語数の領域を確保する。

語数は，10 進定数（≧0）で指定する。語数を 0 とした場合，領域は確保しないが，ラベルは有効である。

(4) | DC | 定数［,定数］… |

DC 命令は，定数で指定したデータを（連続する）語に格納する。

定数には，10 進定数，16 進定数，文字定数，アドレス定数の 4 種類がある。

定数の種類	書き方	命 令 の 説 明
10 進定数	n	n で指定した 10 進数値を，1 語の 2 進数データとして格納する。ただし，n が−32768～32767 の範囲にないときは，その下位 16 ビットを格納する。
16 進定数	#h	h は 4 けたの 16 進数（16 進数字は 0～9，A～F）とする。h で指定した 16 進数値を 1 語の 2 進数データとして格納する（0000 ≦ h ≦ FFFF）。
文字定数	'文字列'	文字列の文字数（＞0）分の連続する領域を確保し，最初の文字は第 1 語の下位 8 ビットに，2 番目の文字は第 2 語の下位 8 ビットに，…と順次文字データとして格納する。各語の上位 8 ビットには 0 のビットが入る。 文字列には，間隔及び任意の図形文字を書くことができる。ただし，アポストロフィ（'）は 2 個続けて書く。
アドレス定数	ラベル	ラベルに対応するアドレスを 1 語の 2 進数データとして格納する。

2.4 マクロ命令

マクロ命令は，あらかじめ定義された命令群とオペランドの情報によって，目的の機能を果たす命令群を生成する（語数は不定）。

(1) | IN | 入力領域,入力文字長領域 |

IN 命令は，あらかじめ割り当てた入力装置から，1 レコードの文字データを読み込む。

入力領域は，256 語長の作業域のラベルであり，この領域の先頭から，1 文字を 1 語に対応させて順次入力される。レコードの区切り符号（キーボード入力の復帰符号など）は，格納しない。格納の形式は，DC 命令の文字定数と同じである。入力データが 256 文字に満たない場合，入力領域の残りの部分は実行前のデータを保持する。入力データが 256 文字を超える場合，以降の文字は無視される。

入力文字長領域は，1 語長の領域のラベルであり，入力された文字の長さ（≧0）が 2 進数で格納される。ファイルの終わり（end of file）を検出した場合は，−1 が格納される。

IN 命令を実行すると，GR の内容は保存されるが，FR の内容は不定となる。

(2) | OUT | 出力領域,出力文字長領域 |

OUT 命令は，あらかじめ割り当てた出力装置に，文字データを，1 レコードとして書き出す。

出力領域は，出力しようとするデータが 1 文字 1 語で格納されている領域のラベルである。格納の形式は，DC 命令の文字定数と同じであるが，上位 8 ビットは，OS が無視するので 0 でなくてもよい。

出力文字長領域は，1 語長の領域のラベルであり，出力しようとする文字の長さ（≧0）を 2 進数で格納しておく。

OUT 命令を実行すると，GR の内容は保存されるが，FR の内容は不定となる。

(3) RPUSH

RPUSH 命令は，GR の内容を，GR1，GR2，…，GR7 の順序でスタックに格納する。

(4) RPOP

RPOP 命令は，スタックの内容を順次取り出し，GR7，GR6，…，GR1 の順序で GR に格納する。

2.5 機械語命令

機械語命令のオペランドは，次の形式で記述する。

r, r1, r2	GR は，記号 GR0 ～ GR7 で指定する。
x	指標レジスタとして用いる GR は，記号 GR1 ～ GR7 で指定する。
adr	アドレスは，10 進定数，16 進定数，アドレス定数又はリテラルで指定する。 リテラルは，一つの 10 進定数，16 進定数又は文字定数の前に等号 (=) を付けて記述する。CASL Ⅱは，等号の後の定数をオペランドとする DC 命令を生成し，そのアドレスを adr の値とする。

2.6 その他

(1) アセンブラによって生成される命令語や領域の相対位置は，アセンブラ言語での記述順序とする。ただし，リテラルから生成される DC 命令は，END 命令の直前にまとめて配置される。

(2) 生成された命令語，領域は，主記憶上で連続した領域を占める。

3. プログラム実行の手引

3.1 OS

プログラムの実行に関して，次の取決めがある。

(1) アセンブラは，未定義ラベル（オペランド欄に記述されたラベルのうち，そのプログラム内で定義されていないラベル）を，他のプログラムの入口名（START 命令のラベル）と解釈する。この場合，アセンブラはアドレスの決定を保留し，その決定を OS に任せる。OS は，実行に先立って他のプログラムの入口名との連係処理を行いアドレスを決定する（プログラムの連係）。

(2) プログラムは，OS によって起動される。プログラムがロードされる主記憶の領域は不定とするが，プログラム中のラベルに対応するアドレス値は，OS によって実アドレスに補正されるものとする。

(3) プログラムの起動時に，OS はプログラム用に十分な容量のスタック領域を確保し，その最後のアドレスに 1 を加算した値を SP に設定する。

(4) OS は，CALL 命令でプログラムに制御を渡す。プログラムを終了し OS に制御を戻すときは，RET 命令を使用する。

(5) IN 命令に対応する入力装置，OUT 命令に対応する出力装置の割当ては，プログラムの実行に先立って利用者が行う。

(6) OS は，入出力装置や媒体による入出力手続の違いを吸収し，システムでの標準の形式及び手続（異常処理を含む）で入出力を行う。したがって，IN，OUT 命令では，入出力装置の違いを意識する必要はない。

3.2 未定義事項

プログラムの実行等に関し，この仕様で定義しない事項は，処理系によるものとする。

〔 メ モ 用 紙 〕

〔 メ モ 用 紙 〕

10. 答案用紙の記入に当たっては，次の指示に従ってください。

(1) HB の黒鉛筆又はシャープペンシルを使用してください。訂正の場合は，あとが残らないように消しゴムできれいに消し，消しくずを残さないでください。

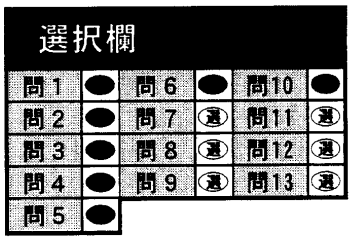
(2) 答案用紙は光学式読取り装置で処理しますので，答案用紙の**マークの記入方法**のとおりマークしてください。

(3) **受験番号欄**に，**受験番号**を記入及びマークしてください。正しくマークされていない場合，答案用紙のマークの記入方法のとおりマークされていない場合は，採点されません。

(4) **生年月日欄**に，受験票に印字されているとおりの**生年月日**を記入及びマークしてください。正しくマークされていない場合は，採点されないことがあります。

(5) **選択した問題**については，次の例に従って，**選択欄**の**問題番号**の (選) をマークしてください。マークがない場合は，採点の対象になりません。

〔**問6**と**問10**を選択した場合の例〕



(6) 解答は，次の例題にならって，解答欄にマークしてください。

〔例題〕　次の［　　］に入れる正しい答えを，解答群の中から選べ。

春の情報処理技術者試験は，［ a ］月に実施される。

解答群

ア　2　　イ　3　　ウ　4　　エ　5

正しい答えは“ウ　4”ですから，次のようにマークしてください。

例題	a	ア　イ　●　エ

11. 試験終了後，この問題冊子は持ち帰ることができます。

12. 答案用紙は，白紙であっても提出してください。

13. 試験時間中にトイレへ行きたくなったり，気分が悪くなったりした場合は，手を挙げて監督員に合図してください。