平成20年度　秋期
基本情報技術者
午後　問題

試験時間	13:00 ～ 15:30（2時間30分）

注意事項

1. 試験開始及び終了は，監督員の時計が基準です。監督員の指示に従ってください。
2. 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。
3. この注意事項は，問題冊子の**裏表紙**に続きます。必ず読んでください。
4. **答案用紙への受験番号などの記入は，試験開始の合図があってから始めてください。**
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

6. 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) B又は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〕

選択欄					
問1	●	問6	●	問10	●
問2	●	問7	(選)	問11	(選)
問3	●	問8	(選)	問12	(選)
問4	●	問9	(選)	問13	(選)
問5	●				

   (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。

秋の情報処理技術者試験は，［ a ］月に実施される。

解答群

ア　8　　イ　9　　ウ　10　　エ　11

正しい答えは“ウ　10”ですから，次のようにマークしてください。

例題	a	(ア) (イ) ● (エ)

裏表紙の注意事項も，必ず読んでください。

C　COBOL　Java　アセンブラ

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　通信ネットワークに関する次の記述を読んで，設問1，2に答えよ。

A社は，本社及び三つの支店から成り，合計4か所の拠点を結ぶ通信ネットワークを構築している。支店同士は直接通信できず，本社に設置されたルータを経由して通信する。ネットワーク構成を図1に，支店間の通信経路を図2に示す。

(凡例)
○：アクセスポイント
↔：通信経路
w，x，y，z：拠点とその拠点に対応したアクセスポイントとの接続

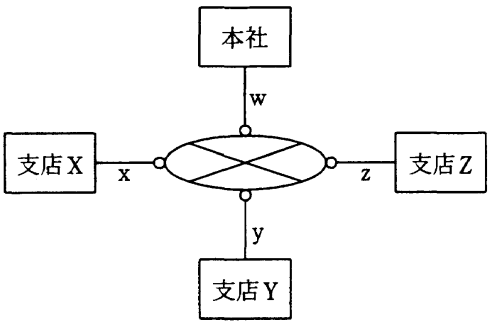


図1　ネットワーク構成

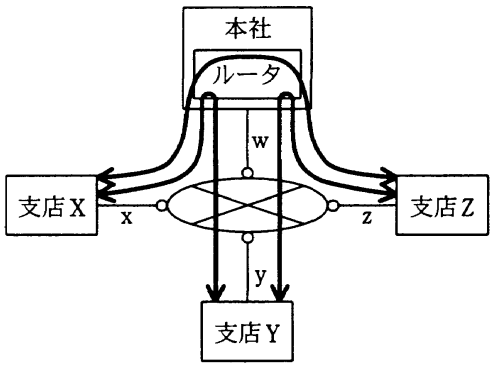


図2　支店間の通信経路

A社は，通信ネットワーク上にIP電話による内線網を構築し，拠点間の通話を行っている。ピーク時における各拠点間の同時通話数は，表のとおりである。

表　ピーク時における各拠点間の同時通話数

			支店Z
		支店Y	10
	支店X	10	10
本社	20	20	20

なお，1通話が使用するビット速度（帯域幅）は，データ圧縮をするので送話，受話ともに20kビット／秒であり，通話接続の制御に伴う通信量は無視できるものとする。

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。

支店X，Y間の通話には，支店Xから支店Yに向かう音声と，支店Yから支店Xに向かう音声による通信が発生する。支店間の通信は本社に設置されたルータを必ず経由するので，これらの通信は本社のアクセスポイントとルータとの間wを往復する。したがって，支店X，Y間の通話について，wのトラフィックは，支店Xとアクセスポイントとの間xの［ a ］倍である。

IP電話のピーク時のトラフィックは，wでは，上り下りとも2,400kビット／秒であり，xでは，上り下りとも［ b ］kビット／秒である。ただし，拠点からアクセスポイントに向かう通信を上り，アクセスポイントから拠点に向かう通信を下りとする。

aに関する解答群

ア　1/4　　イ　1/2　　ウ　2　　エ　4

bに関する解答群

ア　400　　イ　800　　ウ　1,200　　エ　2,400

設問2　A社は支店間の通信形態を見直して，本社のルータを経由せずに，支店同士が直接通信できるようにする。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。

通信形態の見直しによって，想定されるピーク時のIP電話によるトラフィックは，本社のアクセスポイントとルータとの間wでは，上り下りとも［ c ］kビット／秒となる。各支店とアクセスポイントとの間（x，y，z）のトラフィックは，見直し前と比べて［ d ］である。

cに関する解答群

ア　400　　イ　1,200　　ウ　1,800　　エ　2,400

dに関する解答群

ア　1/4倍　　イ　1/2倍　　ウ　同じ　　エ　2倍　　オ　4倍

問2　次の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〕

文字型の1次元配列 Str に格納されている文字列を，整形して出力する副プログラム PrintOut である。

(1) 文字列は，英数字，記号及び改行文字で構成される。

(2) 文字列は，文字型の1次元配列の各要素に1文字ずつ格納され，文字列の最後の要素の次の要素には，システム定数 EOS が格納されている。

(3) PrintOut の仕様は，次のとおりである。出力結果の例を図に，引数の仕様を表1に示す。

① 指定した文字数（引数 Margin）だけ，左側に余白を設ける。

② 指定した1行の最大文字数（引数 MaxC）で改行する。

③ 指定した1ページの最大行数（引数 MaxL）で改ページする。

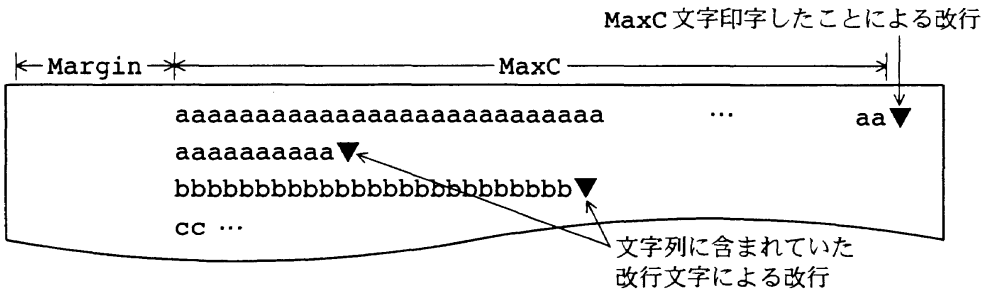


図　出力結果の例

(4) PrintOut は，次の副プログラム及びシステム関数を用いる。各副プログラム及びシステム関数の引数の仕様を表2～4に示す。

① 副プログラム GetPosition：文字列の指定した位置から行末となる文字位置を探索し，その位置を引数 EndP に返す。行末の条件は，EOS が見つかったとき，改行文字が見つかったとき，又は MaxC ＋ 1 文字に達したときのいずれかである。

② 副プログラム PutLine：指定された文字列を出力し，最後に改行を行う。

③ システム関数 PutChr：指定された1文字を出力する。PutChr(NL) で印字位置は次の行の左端に移動し，PutChr(FF) で印字位置は次ページの左上端に移動する。ここで，NL は改行文字を，FF は改ページ文字を表すシステム定数である。

(5) 配列の添字は0から始まる。

表１　PrintOut の引数の仕様

引数	データ型	入力／出力	意味
Str[]	文字型	入力	整形対象の文字列を格納した１次元配列
Margin	整数型	入力	左側の余白の文字数 Margin≧0，1≦（Margin＋MaxC）≦100
MaxC	整数型	入力	1行に出力する最大文字数 MaxC≧1，1≦（Margin＋MaxC）≦100
MaxL	整数型	入力	１ページに出力する最大行数 MaxL≧1

表２　GetPosition の引数の仕様

引数	データ型	入力／出力	意味
Str[]	文字型	入力	整形対象の文字列を格納した１次元配列
StartP	整数型	入力	探索の開始位置
MaxC	整数型	入力	1行に出力する最大文字数
EndP	整数型	出力	探索の結果，得られた行末位置

表３　PutLine の引数の仕様

引数	データ型	入力／出力	意味
Line[]	文字型	入力	出力対象の文字列を格納した１次元配列

表４　PutChr の引数の仕様

引数	データ型	入力／出力	意味
Chr	文字型	入力	出力対象の１文字

〔プログラム〕

```
○PrintOut(文字型: Str[], 整数型: Margin, 整数型: MaxC, 整数型: MaxL)
○文字型: Line[101]
○整数型: StartP, LineC, MarginP, CurrentP, EndP
・StartP ← 0
・LineC ← 0
■ MarginP: 0, MarginP < Margin, 1
  ・Line[MarginP] ← " "  /* 空白文字 */
■

■ Str[StartP] ≠ EOS        /* EOS: 文字列の終わりを表すシステム定数 */
  ・GetPosition(Str[], StartP, MaxC, EndP)
  ■ [ a ]
    ・Line[CurrentP] ← Str[StartP]
    ・StartP ← StartP + 1
  ■
  ・Line[CurrentP] ← EOS
  ・PutLine(Line[])
  ・LineC ← LineC + 1
  ▲ LineC ≧ MaxL
    ・PutChr(FF)           /* FF: 改ページを表すシステム定数 */
    ・LineC ← 0
  ▼
  ▲ [ b ]
    ・StartP ← StartP + 1
  ▼
■

○GetPosition(文字型: Str[], 整数型: StartP, 整数型: MaxC, 整数型: EndP)
・EndP ← StartP
■ (Str[EndP] ≠ EOS) and (Str[EndP] ≠ NL) and ( [ c ] )
  ・EndP ← EndP + 1
■

○PutLine(文字型: Line[])
○整数型: CurrentP
■ [ d ]
  ・PutChr(Line[CurrentP])
■
・PutChr(NL)                  /* NL: 改行を表すシステム定数 */
```

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　CurrentP: 0, CurrentP ＜ MaxC, 1

イ　CurrentP: 0, StartP ＜ EndP, 1

ウ　CurrentP: Margin, CurrentP ＜ MaxC, 1

エ　CurrentP: Margin, StartP ＜ EndP, 1

bに関する解答群

ア　Str[StartP] ＝ EOS　　イ　Str[StartP] ＝ NL

ウ　Str[StartP] ≠ EOS　　エ　Str[StartP] ≠ NL

cに関する解答群

ア　EndP ＜ (StartP ＋ MaxC)　　イ　EndP ＜ MaxC

ウ　StartP ＜ (EndP ＋ MaxC)　　エ　StartP ＜ MaxC

dに関する解答群

ア　CurrentP: 0, Line[CurrentP] ＝ EOS, 1

イ　CurrentP: 0, Line[CurrentP] ＝ NL, 1

ウ　CurrentP: 0, Line[CurrentP] ≠ EOS, 1

エ　CurrentP: 0, Line[CurrentP] ≠ NL, 1

問3　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。

B社では，社員に関連する関係データベースを構築し，各部署からの問合せに対応している。B社の関係データベースの構造の一部を図に示す。

社員表（社員コード，氏名，入社年月日，部署コード，役職コード，退職年月日，生年月日，住所，電話番号）
部署表（部署コード，部署名）
役職表（役職コード，役職名）
研修実施表（年度，研修コード，研修名，研修開始年月日，研修終了年月日，研修日数）
受講表（年度，研修コード，社員コード，受講日数）

注　下線はキー項目を表す。

図　B社の関係データベースの構造（一部）

〔各表の概要〕

(1) 社員表は，社員（退職者を含む）の情報を管理する。在職中の社員の退職年月日はNULLとする。

(2) 部署表は，B社の部署名を管理する。

(3) 役職表は，B社の役職名を管理する。

(4) 研修実施表は，年度ごとにB社が実施した研修名と研修期間を管理する。

(5) 受講表は，各社員が受講した研修とその日数（受講日数）を管理する。社員は，本人が必要と思う研修だけに申し込み，受講する。受講予定の社員が受講しなかった場合，その研修の受講日数の値は0となる。

(6) B社で規定する年度は，その年の4月1日から翌年の3月31日までである。
2007年度は，2007年4月1日から2008年3月31日までとなる。

設問1　入社2年目の社員が，入社以来1日でも受講した研修にどのようなものがあるかを知りたい。

次のSQL文中の［　　］に入れる正しい答えを，解答群の中から三つ選べ。ここで，入社2年目の社員とは，入社した年度が2007年度で，かつ在職中の社員とする。

```
SELECT DISTINCT 研修名 FROM 受講表, 研修実施表, 社員表
   WHERE 受講表.社員コード = 社員表.社員コード AND
         [　　]  AND
         [　　]  AND
         [　　]  AND
         退職年月日 IS NULL AND
         入社年月日 BETWEEN '2007-04-01' AND '2008-03-31'
```

解答群

ア　研修実施表.研修コード = 受講表.研修コード

イ　研修実施表.年度 = '2007'

ウ　研修実施表.年度 = 受講表.年度

エ　研修日数 > 0

オ　受講日数 > 0

カ　受講表.年度 = '2007'

設問2　B社では，研修を受講した社員に対して，本人の受講実績だけが照会できるSQL文が提供されている。しかし，役職名の中に“部長”という文字列がある社員（“部長”，“副部長”，“部長代理”など）は，自分の受講実績だけでなく，同じ所属部署にいるすべての社員の受講実績を照会できるようにしたい。この照会ができる社員を選択するSQL文の説明として［　　］に入れる正しい答えを，解答群の中から選べ。

役職表の役職名に“部長”という文字列を含むすべての役職は［ a ］として検索する。同じ所属部署の社員を抽出するには，次のとおりにする。

受講表には［ b ］という項目があるが，これだけではその社員が同じ所属部署であるかどうか分からない。同じ所属部署であるかどうかを知るためには［ c ］という項目が必要になる。したがって，受講表の項目［ b ］を用いて［ d ］を結合し，［ c ］の値を取り出せばよい。

aに関する解答群

ア　SELECT 役職コード FROM 役職表 WHERE 役職名 = '部長'

イ　SELECT 役職コード FROM 役職表
　　WHERE 役職名 = '部長' OR 役職名 = '副部長' OR 役職名 = '部長代理'

ウ　SELECT 役職コード FROM 役職表
　　WHERE 役職名 IN ('部長', '副部長', '部長代理')

エ　SELECT 役職コード FROM 役職表 WHERE 役職名 LIKE '%部長%'

オ　SELECT 役職コード FROM 役職表 WHERE 役職名 LIKE '部長%'

b～dに関する解答群

ア　研修コード　　イ　研修実施表

ウ　社員コード　　エ　社員表

オ　部署コード　　カ　部署表

キ　役職コード　　ク　役職表

問4　次のアルゴリズムの説明を読んで，設問1，2に答えよ。

〔アルゴリズムの説明〕

N個（N＞1）の地点から構成されるグラフにおいて，出発地点からそれ以外の地点までの最短距離を求めるアルゴリズムShortestLengthである。

(1) 図1に示す地点数N＝6のグラフを例として，ShortestLengthを説明する。図1において，丸は地点を，矢印の向きは進行方向を，矢印に付けた数字は地点間の距離を表す。

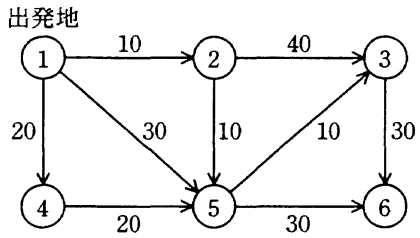


図1　グラフの例（地点数N＝6）

(2) ShortestLengthでは，次の配列を用いる。要素番号は1，2，…，Nで，地点と対応している。配列の添字は1から始まる。

Dt[i][j]：地点iから地点jまでの距離が格納されているN×N個の要素からなる配列。地点iから地点jまでの直接の経路がない場合，進行方向が逆向きの場合及びDt[i][i]の場合には∞（最大値を表す定数）が格納されている。図1の例では，Dt[i][j]の一部は次のとおりである。

Dt[1][1]＝∞，Dt[1][2]＝10，

Dt[1][3]＝∞，Dt[2][1]＝∞

Sd[i]：出発地である地点1から地点iまでの仮の最短距離（以下，仮最短距離という）を格納する配列。初期設定ではすべての要素に∞を格納する。

Pe[i]：最短距離を求める過程で処理済となった地点を識別するための配列。初期設定ではすべてfalseに設定する。Pe[i]がtrueとなったとき，Sd[i]には地点1から地点iまでの最短距離が求まっている。すべての要素がtrueになると処理は終了する。

(3) 最短距離を求める手順を，図1の例を使って示す。

① 出発地は地点1なので，Pe[1]にtrueを設定し，地点1から直接つながる地点2，4，5までの距離Dt[1][2]，Dt[1][4]，Dt[1][5]をそれぞれSd[2]，Sd[4]，Sd[5]に設定する。その結果，Sd[2]＝10，Sd[4]＝20，Sd[5]＝30

となる。この結果を図2に示す。地点iの上又は下にある[ ]内の数値は，地点1から地点iまでの仮最短距離Sd[i]を表し，網掛けの地点は処理済（Pe[i]＝true）を表す。

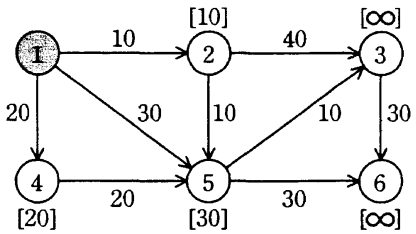


図2　手順①の結果

② 未処理の地点のうち，地点1からの仮最短距離が最も短い地点2（2はPe[k] ＝ falseかつSd[k]が最小となる添字kの値）を選択し，Pe[2]をtrueにする。この時点で，Sd[2]＝10が地点1から地点2の間の最短距離として確定する。

次に，地点2から直接行ける地点3と地点5の仮最短距離Sd[i]を更新する。ただし，Sd[i]の値が小さくならない場合は置き換えない。

（更新前）　　（更新後）

Sd[3]＝∞ → Sd[3]＝Sd[2]＋Dt[2][3]＝50

Sd[5]＝30 → Sd[5]＝Sd[2]＋Dt[2][5]＝20

このとき，地点k（k＝2）を経由する地点i（i＝3，5）の仮最短距離Sd[i]を更新するための代入文を擬似言語で書くと，次の〔プログラムの一部〕のとおりになる。ここで，関数minは二つの引数のうち，小さい方の値を返すシステム関数である。

〔プログラムの一部〕

・Sd[i] ← min(Sd[i], [a])

この結果を図3に示す。

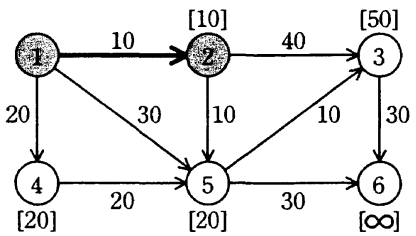


図3　手順②の結果

③　未処理の地点のうち，仮最短距離が最も短い地点は，地点4と地点5の二つである。このように複数ある場合は，要素番号の小さい地点4を選択し，処理済とする。この時点で，Sd[4]が地点1から地点4の間の最短距離として確定する。

次に，地点4から直接行ける地点の仮最短距離を更新する。

Sd[4]＋Dt[4][5]＝40となり，現在のSd[5]の値より大きいので，更新しない。この結果を図4に示す。

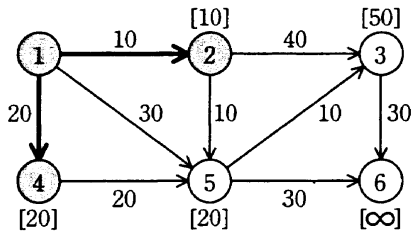


図4　手順③の結果

④　未処理の地点のうち，仮最短距離が最も短い地点5を選択し，処理済とする。この時点で，Sd[5]が地点1から地点5の間の最短距離として確定する。

次に，地点5から直接行ける地点の仮最短距離を更新する。

Sd[3]＝50 → Sd[3]＝Sd[5]＋Dt[5][3]＝30

Sd[6]＝∞ → Sd[6]＝Sd[5]＋Dt[5][6]＝50

この結果を図5に示す。

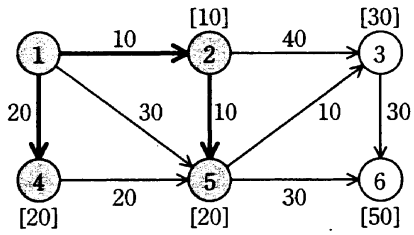


図5　手順④の結果

⑤　未処理の地点のうち，仮最短距離が最も短い地点3を選択し，処理済とする。この時点で，Sd[3]が地点1から地点3の間の最短距離として確定する。

次に，地点3から直接行ける地点6の仮最短距離を更新する。

手順③と同様に計算し，地点6の仮最短距離Sd[6]の値は［　b　］となる。

この結果を図6に示す。

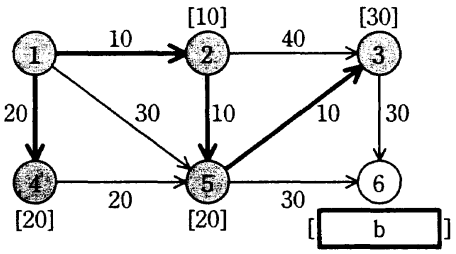


図6　手順⑤の結果

⑥　未処理の地点のうち，仮最短距離が最も短い地点6を選択し，処理済とする。

この時点で，Sd[6]が地点1から地点6の間の最短距離として確定する。

すべての地点が処理済となったので終了となる。この結果を図7に示す。

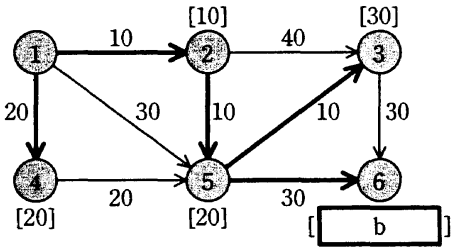


図7　手順⑥の結果

設問1　アルゴリズム ShortestLength の説明中の［　　］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　Sd[i] + Dt[i][k]　　イ　Sd[i] + Dt[k][i]

ウ　Sd[k] + Dt[i][k]　　エ　Sd[k] + Dt[k][i]

bに関する解答群

ア　30　　イ　40　　ウ　50　　エ　60

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。

図8のグラフの場合，出発地を地点1としてアルゴリズム ShortestLength を実行したとき，最短距離が確定する順番は，次のとおりになる。

地点1，［　c　］，地点6

また，配列Sdの要素Sd[3]，Sd[5]及びSd[6]の値は，それぞれ［　d　］，［　e　］及び［　f　］となる。

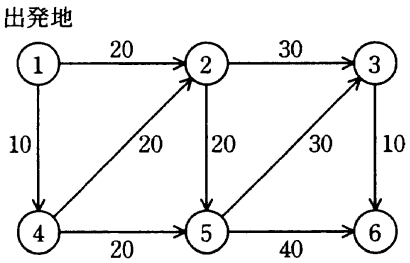


図8　グラフ

cに関する解答群

ア　地点2，地点4，地点3，地点5

イ　地点2，地点4，地点5，地点3

ウ　地点4，地点2，地点3，地点5

エ　地点4，地点2，地点5，地点3

d～fに関する解答群

ア　30	イ　40	ウ　50	エ　60
オ　70	カ　80	キ　90	ク　100

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。

C社では，ディジタル化した楽曲を会員がダウンロードできる楽曲ダウンロードシステム（以下，システムという）を提供している。

〔システムの概要〕

(1) 楽曲の情報は，楽曲番号をキーとして楽曲管理ファイルに記録される。楽曲番号は，それぞれの楽曲データに割り当てられる一意な番号である。楽曲管理ファイルのレコード様式を図1に示す。

楽曲番号	曲名	演奏者名	演奏時間	制作会社	ジャンル	楽曲データの大きさ	楽曲データの格納場所

注　下線はキー項目を表す。

図1　楽曲管理ファイルのレコード様式

(2) 会員の情報は，それぞれの会員に一意になるように割り当てた会員番号をキーとして，会員ファイルに記録される。会員ファイルには，会員番号とともに，暗号化したパスワード，氏名，入会日が記録される。会員ファイルのレコード様式を図2に示す。

会員番号	暗号化したパスワード	氏名	入会日

注　下線はキー項目を表す。

図2　会員ファイルのレコード様式

(3) 会員は，システムにログインするために会員番号とパスワードを入力する。

(4) システムは，(3)で入力された会員番号をキーとして会員ファイルを検索し，会員番号が一致したレコードの暗号化したパスワードを取り出す。暗号化したパスワードと，入力されたパスワードを暗号化した結果が一致した場合は，ログインを許可し，楽曲検索画面を表示する。一致しない場合は，ログイン不許可のメッセージを表示して終了する。

(5) 楽曲検索画面には，曲名からジャンルまでの楽曲管理ファイルの項目（以下，検索項目という）が表示される。ログインした会員は，楽曲の検索条件として，一つ

以上の検索項目に対する検索内容を入力する。複数の検索条件が指定されたときは，入力されたすべての条件を AND で接続した条件による検索が行われる。検索が不要の場合は，ログアウトを選択する。

(6) システムは，(5) で入力された検索内容を基に，楽曲管理ファイルを検索する。条件に一致したすべてのレコードを抽出し，楽曲番号，曲名，演奏者名を楽曲選択画面に表示する。条件に一致するレコードがない場合は，楽曲検索画面にその旨のメッセージを表示する。

(7) (6) で表示された楽曲選択画面で会員が曲名を一つ選択すると，その楽曲のダウンロードがすぐに始まる。再検索を選択すると，楽曲検索画面に戻る。

(8) ダウンロードが完了したらダウンロードした日時をダウンロード実績ファイルに格納し，ダウンロード完了のメッセージを表示して (5) に戻る。ダウンロード実績ファイルのレコード様式を図 3 に示す。

会員番号	ダウンロード日付	ダウンロード時刻	楽曲番号

注　下線はキー項目を表す。

図3　ダウンロード実績ファイルのレコード様式

(9) システムの画面遷移を図 4 に，モジュール構造を図 5 に示す。

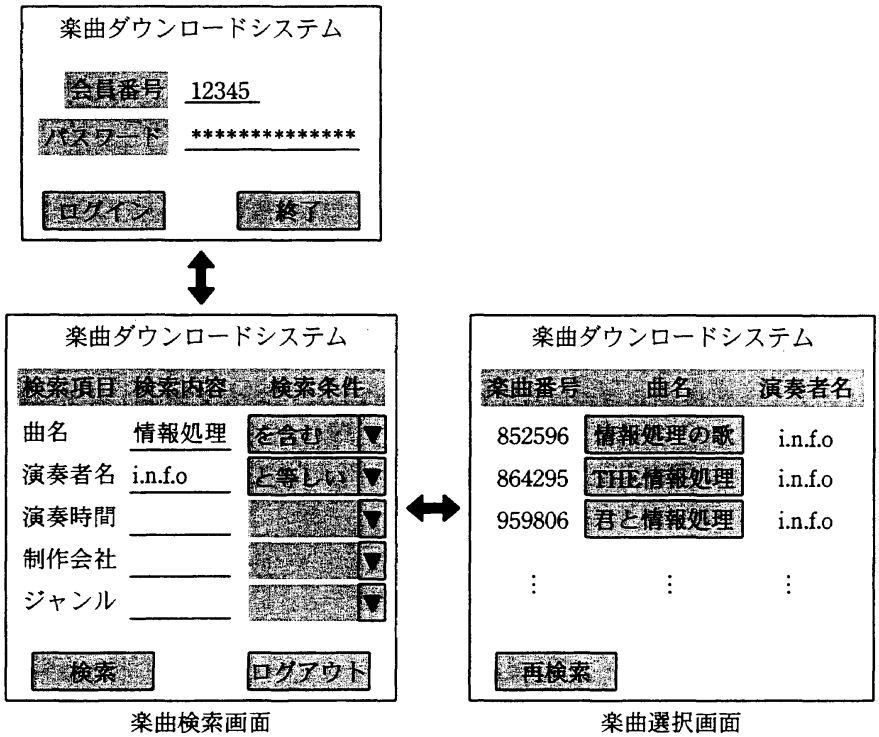


図4　システムの画面遷移

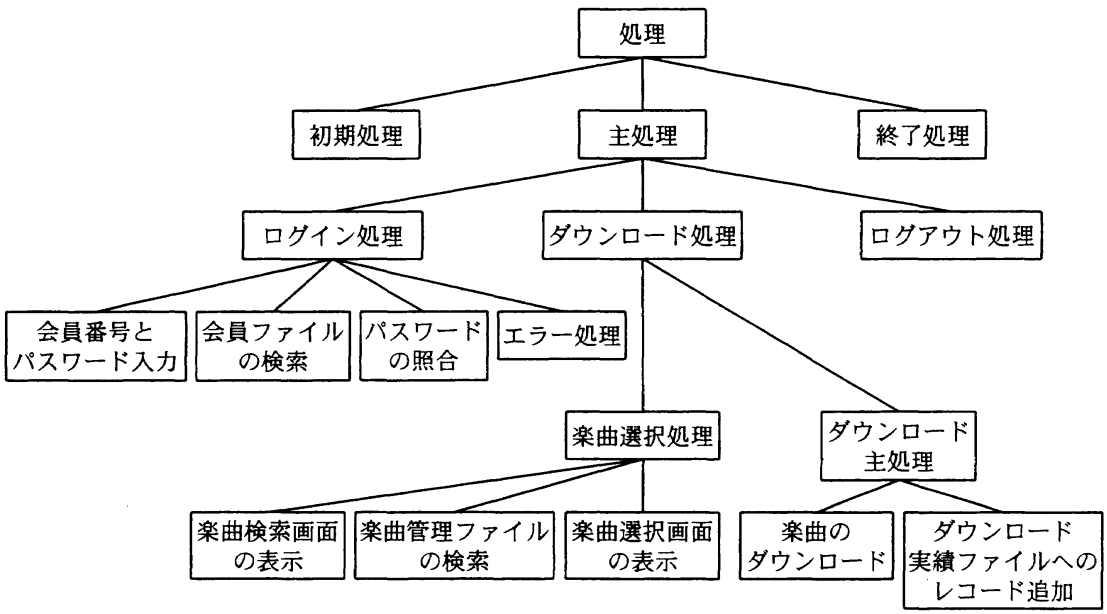


図5　システムのモジュール構造

設問1　会員が1日にダウンロードできる楽曲の件数を，システムで定めた許可件数以下に制限する機能を追加する。次に示すログインした会員の当日のダウンロード件数を求める手順に関する記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

現在日付をシステムから取り出す。次にログインした会員の［　a　］と現在日付の二つをキーとしてダウンロード実績ファイルを検索し，レコードの件数を数える。ログイン後ダウンロード実行前までに日付が変わった場合，新しい日付でダウンロード件数を求める必要がある。したがって，このダウンロード件数を求めるモジュールは，［　b　］モジュールの直前に実行する必要がある。

aに関する解答群

ア　会員番号　　イ　楽曲データの大きさ　　ウ　楽曲番号
エ　ダウンロード日付　　オ　入会日　　カ　パスワード

bに関する解答群

ア　会員ファイルの検索　　イ　楽曲管理ファイルの検索
ウ　楽曲検索画面の表示　　エ　楽曲選択画面の表示
オ　楽曲のダウンロード

設問2　楽曲選択画面の表示項目として，その会員が既にダウンロードしたことのある楽曲については，最後のダウンロード日付を追加する。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。

会員の入力した検索内容で楽曲管理ファイルを検索し，条件に一致するすべてのレコードを抽出する。抽出したレコードの楽曲番号が，その会員が既にダウンロード済の楽曲のものであるかどうかは，ダウンロード実績ファイルを［　c　］で検索すれば判明する。このダウンロード実績ファイルに対する検索では，楽曲管理ファイルから抽出した1件のレコードに対して，その会員のダウンロード実績の件数は［　d　］である。

cに関する解答群

ア　会員番号

イ　会員番号と楽曲番号

ウ　会員番号と楽曲番号とダウンロード日付

エ　会員番号とダウンロード日付

オ　楽曲番号

カ　楽曲番号とダウンロード日付

キ　ダウンロード日付

dに関する解答群

ア　0件　　イ　0件以上　　ウ　0件又は1件

エ　1件　　オ　2件以上

設問3　入会直後の会員のダウンロード状況を把握するために，全会員の入会後1週間におけるジャンル別のダウンロード件数表を作成することにした。ジャンル別ダウンロード件数表の例を図6に示す。また，ジャンル別ダウンロード件数表の作成の流れを図7に，各処理の説明を表に示す。表中の［　　］に入れる正しい答えを，解答群の中から選べ。

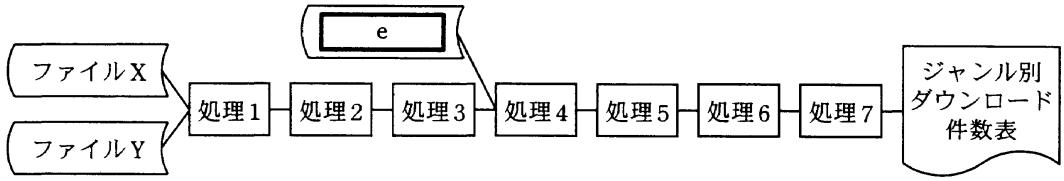
なお，ジャンル別ダウンロード件数表には，図6のようにジャンル別のダウンロード件数とその楽曲数が表示されるものとする。

ジャンル別ダウンロード件数表

2008-10-19

ジャンル	ダウンロード件数	楽曲数
クラシック	23,456	54
ジャズ	4,012	130
ポップス	123,456	434
演歌	34,567	120
民謡	7,410	87
⋮	⋮	⋮

図6　ジャンル別ダウンロード件数表の例



注　ファイルX, Yは，楽曲管理ファイル，会員ファイル又はダウンロード実績ファイルのいずれかを表す。
なお，各処理間の中間ファイルは省略している。

図7　ジャンル別ダウンロード件数表の作成の流れ

表　各処理の説明

処理	種類	内容
処理1	突合せ	ファイルXとファイルYのレコードを，会員番号をキーとして突き合わせる。キーが一致した場合，ダウンロード日付がその会員の入会後1週間以内のとき，ファイルXとファイルYのレコードから必要な項目の値を取り出し，それらを出力する。この処理をすべての会員に対して実施する。
処理2	整列	処理1の結果を読み込み，楽曲番号をキーとして並べ替えて出力する。
処理3	集計	処理2の結果を読み込み，同一の楽曲番号のレコードの件数を集計し，楽曲番号とそのレコードの集計件数を中間ファイルに出力する。
処理4	突合せ	[e] と中間ファイルのレコードを，楽曲番号をキーとして突き合わせる。キーが一致した場合，読み込んだ [e] と中間ファイルのレコードから必要な項目の値を取り出し，それらを出力する。この処理をすべての楽曲に対して実施する。
処理5	整列	処理4の結果を読み込み，[f] をキーとして並べ替えて出力する。
処理6	集計	処理5の結果を読み込み，ジャンルごとにレコードの件数を集計して楽曲数を求める。処理3で出力した集計件数を集計してダウンロード件数を求める。ジャンルとダウンロード件数と楽曲数を出力する。
処理7	作表	処理6の結果を読み込み，ジャンル別ダウンロード件数表を出力する。

eに関する解答群

ア　会員ファイル　　イ　楽曲管理ファイル

ウ　ダウンロード実績ファイル

fに関する解答群

ア　会員番号　　イ　楽曲データの大きさ　　ウ　楽曲番号

エ　ジャンル　　オ　ダウンロード件数

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，設問に答えよ。

〔プログラムの説明〕

金額を表すときのように，整数を 3 けた区切り形式の文字列に変換する関数 convert である。

(1) 次のルールに基づいて変換を行う。

① 整数値が負の場合，先頭にマイナス符号を付ける。

② 数値の下位から3けたごとにコンマを挿入する。

変換例を表に示す。

表　変換例

整数	3けた区切り形式の文字列
1234567	1,234,567
-57482	-57,482
63	63
-999999	-999,999

(2) プログラム中で定義されている関数の仕様は，次のとおりである。

```
void convert(long num, char str[]);
```

引数：整数 num，変換後の文字列 str[]

返却値：なし

機能：整数 num を3けた区切り形式の文字列に変換して str[] に先頭から格納する。str[] には変換後の文字列を格納するのに十分な領域が確保されているものとする。また，整数 num は9けた以下とする。

〔プログラム〕

```
void convert(long, char[]);

void convert(long num, char str[]){
   int minus = 0, i = 0, j = 0;
   char table[] = "0123456789";
   char tmp;

   if(num < 0){
      minus = 1;
      num = -num;
   }

   do{
      str[j++] = table[num % 10];    /* 数値の下位から順に文字に変換 */
      num [   a   ];
      i++;
      if(   b   == 0 && num != 0){
         str[j++] = ',';
      }
   }while(num != 0);

   if(minus != 0){
      str[j++] = '-';
   }
   str[j--] = '\0';

   for(i = 0;   c   ){              /* 順序を逆にする。 */
      tmp = str[i];
      str[i] = str[j];
      str[j] = tmp;
   }
}
```

C

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　`%= 10`　　イ　`*= -1`

ウ　`*= -10`　　エ　`*= 10`

オ　`/= 10`

C

bに関する解答群

ア　`(i + 1) % 3`　　イ　`(i + 2) % 3`

ウ　`(j + 1) % 3`　　エ　`(j + 2) % 3`

オ　`i % 3`　　カ　`j % 3`

cに関する解答群

ア　`i != j; i++`　　イ　`i != j; i++, j--`

ウ　`i < j; i++`　　エ　`i < j; i++, j--`

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1，2に答えよ。

〔プログラムの説明〕

ある事業所では，環境活動の一環としてコピー用紙の削減を目標にかかげ，部署ごとの使用状況を月ごとに集計し，公開することにした。この事業所では，両面コピー機能がある1台のコピー機をすべての部署で共用している。このプログラムは，コピー機に記録された稼働情報を集計する。

(1) コピー機の利用者は，部署ごとに配布されたICカードをコピー機にセットする。コピー機はセットされたICカードから利用部署を読み取り，内部に記録する。

(2) 利用実績ファイルCOPY-FILEは，コピー機に記録された情報から集計月分を抽出した，図1に示すレコード様式の順ファイルである。

利用日 (8けた)	利用部署 (1けた)	使用枚数 (4けた)	両面使用 (1けた)

図1　利用実績ファイルのレコード様式

① コピー機の利用1回に対して，1レコードが生成される。

② 利用日には，西暦年が4けた，月及び日がそれぞれ2けたで格納される。

③ 利用部署には，第1営業部，第2営業部，第1開発部，第2開発部，管理部の順に，それぞれ1から5まで割り振られた1けたの部署コードが格納される。

④ 使用枚数には，使用したコピー用紙の枚数が格納される。両面コピー機能を利用して両面に印刷した場合でも1枚と数える。

⑤ 両面コピー機能を利用した場合は両面使用に1が設定され，利用しなかった場合は0が設定される。

(3) コピー用紙の使用量は，1部署当たり合計で毎月3,000枚以下とする。

(4) 集計結果の表示例を図2に示す。

```
  DEPT  ----+----1----+----2----+----3  (x1000)
Sales-1 ************
Sales-2 ********
Dev-1   *******
Dev-2   ****************
Admin   *****
```

図2　集計結果の表示例

① 利用部署の，第1営業部，第2営業部，第1開発部，第2開発部，管理部は，それぞれSales-1，Sales-2，Dev-1，Dev-2，Adminと表示する。

② 部署ごとに月ごとの使用枚数を集計して，100枚を*印一つとした棒グラフで表示する。ただし，100枚に満たない分は切り上げる。

COBOL

〔プログラム〕

(行番号)

```
 1  DATA DIVISION.
 2  FILE SECTION.
 3  FD  COPY-FILE.
 4  01  CP-REC.
 5      02  CP-DATE       PIC X(8).
 6      02  CP-DEPT       PIC 9(1).
 7      02  CP-CNT        PIC 9(4).
 8      02  CP-BOTH       PIC 9(1).
 9  WORKING-STORAGE SECTION.
10  01  TOTAL.
11      02  DEPT-TOTAL    PIC 9(4)  OCCURS 5 VALUE ZERO.
12  77  READ-FLAG         PIC X(1)  VALUE SPACE.
13      88  DATA-EOF      VALUE "E".
14  77  DEPT-CNT          PIC 9(2).
15  77  MARK-CNT          PIC 9(2).
16  77  HEADER            PIC X(47) VALUE
17      "  DEPT  ----+----1----+----2----+----3  (x1000)".
18  01  DEPT              PIC X(40) VALUE
19      "Sales-1 Sales-2 Dev-1   Dev-2   Admin".
20  01  DEPT-HEADER REDEFINES DEPT.
21      02  DEPT-NAME     PIC X(8)  OCCURS 5.
22  77  MARKER            PIC X(30) VALUE ALL "*".
```

```
23  PROCEDURE DIVISION.
24  MAIN-PROC.
25      OPEN INPUT COPY-FILE.
26      PERFORM UNTIL DATA-EOF
27        READ COPY-FILE AT END SET DATA-EOF TO TRUE
28                     NOT AT END PERFORM CNT-PROC
29        END-READ
30      END-PERFORM.
31      CLOSE COPY-FILE.
32      PERFORM PRT-PROC.
33      STOP RUN.
34  CNT-PROC.
35      [          a          ].
36  PRT-PROC.
37      DISPLAY HEADER.
38      PERFORM VARYING DEPT-CNT FROM 1 BY 1 UNTIL DEPT-CNT > 5
39        IF DEPT-TOTAL(DEPT-CNT) NOT = ZERO THEN
40           [          b          ]
41           DISPLAY DEPT-NAME(DEPT-CNT) MARKER(1:MARK-CNT)
42        ELSE
43           DISPLAY DEPT-NAME(DEPT-CNT)
44        END-IF
45      END-PERFORM.
```

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。

解答群

ア　ADD 1 TO DEPT-TOTAL(CP-DEPT)

イ　ADD CP-CNT TO DEPT-TOTAL(CP-DEPT)

ウ　COMPUTE MARK-CNT = (DEPT-TOTAL(DEPT-CNT) + 99) / 100

エ　COMPUTE MARK-CNT = DEPT-TOTAL(DEPT-CNT) / 100

オ　MOVE CP-CNT TO DEPT-TOTAL(CP-DEPT)

カ　MOVE DEPT-TOTAL(DEPT-CNT) TO MARK-CNT

設問2　両面コピー機能を利用した枚数の比率を計算し，50％未満の場合は図3のように棒の右端に“(!)”を表示するように，プログラムを変更する。次の表中の［　　］に入れる正しい答えを，解答群の中から選べ。

```
  DEPT  ----+----1----+----2----+----3  (x1000)
Sales-1 ************
Sales-2 ********(!)
Dev-1   *******
Dev-2   ****************(!)
Admin   *****
```

図3　プログラム変更後の表示例

表　プログラムの変更内容

処置	変更内容
行番号11と12の間に追加	02 DEPT-BOTH PIC 9(4) OCCURS 5 VALUE ZERO. 77 BOTH-RATE PIC 9(3).
［ c ］の間に追加	IF CP-BOTH = 1 THEN ADD CP-CNT TO DEPT-BOTH(CP-DEPT) END-IF.

行番号41を変更	［ d ］
	IF BOTH-RATE >= 50 THEN
	  DISPLAY DEPT-NAME(DEPT-CNT) MARKER(1:MARK-CNT)
	ELSE
	  DISPLAY DEPT-NAME(DEPT-CNT) MARKER(1:MARK-CNT) "(!)"
	END-IF

cに関する解答群

ア　行番号30と31　　イ　行番号31と32

ウ　行番号35と36　　エ　行番号37と38

dに関する解答群

ア
```
COMPUTE BOTH-RATE =
   DEPT-BOTH(DEPT-CNT) * 100 / DEPT-TOTAL(DEPT-CNT)
```

イ
```
COMPUTE BOTH-RATE = DEPT-BOTH(DEPT-CNT) / 100
```

ウ
```
COMPUTE MARK-CNT = DEPT-TOTAL(DEPT-CNT) / 100
```

エ
```
MOVE DEPT-BOTH(DEPT-CNT) TO BOTH-RATE
```

オ
```
MOVE DEPT-TOTAL(DEPT-CNT) TO MARK-CNT
```

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1，2に答えよ。

（Javaプログラムで使用するAPIの説明は，この冊子の末尾を参照してください。）

〔プログラムの説明〕

クラス PhoneBook は電話帳を表すクラスである。一つの名前に複数の電話番号を対応付けることができる。

PhoneBook のメソッドは，次のとおりである。ここで，引数 name は名前，phoneNumber は電話番号を表す。

`void add(String name, String phoneNumber)`

電話帳に名前と電話番号を登録する。このメソッドを複数回呼ぶことによって，一つの名前に複数の電話番号を対応付けることができる。

引数のいずれか又は両方が null のときは，NullPointerException を投げる。また，電話番号は，数字と“#”だけからなる 1 文字以上の文字列とし，誤りがある場合には IllegalArgumentException を投げる。

`Set<String> get(String name)`

引数に指定した名前に対応付けられたすべての電話番号の集合を返す。

`void remove(String name)`

引数に指定した名前とそれに対応付けられているすべての電話番号を削除する。

`void remove(String name, String phoneNumber)`

引数に指定した名前に対応付けられている指定した電話番号を削除する。

クラス PhoneBookTester はテスト用のプログラムである。実行結果を図1に示す。

```
Scott
  031234567#001
```

図1　実行結果

〔プログラム1〕

```
import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;

public class PhoneBook {
   Map<String, Set<String>> book =
                              new HashMap<String, Set<String>>();

   public void add(String name, String phoneNumber) {
      if (name == null || phoneNumber == null) {
         throw new NullPointerException();
      }
      // 数字と“#”だけからなる1文字以上の文字列でなければ例外を投げる。
      if (!phoneNumber.matches("^[0-9#]+$")) {
         throw new IllegalArgumentException();
      }
      Set<String> numbers = book.get(name);
      if (numbers == null) {
         numbers = new HashSet<String>();
      }
      numbers.[        a        ];
      book.[           b          ];
   }

   public Set<String> get(String name) {
      // 指定した名前に対応付けられたすべての電話番号の集合を返す。
      Set<String> numbers = book.get(name);
      Set<String> set = new HashSet<String>();
      if (numbers != null) {
         set.addAll(numbers);
      }
      return set;
   }

   public void remove(String name) {
      book.remove(name);
   }

   public void remove(String name, String phoneNumber) {
      Set<String> numbers = book.get(name);
      if (numbers != null) {
         numbers.[          c          ];
         // 対応付けられた電話番号がなくなれば，電話帳から名前を削除する。
         if (numbers.isEmpty()) {
            book.remove(name);
         }
      }
   }
}
```

〔プログラム2〕

```
public class PhoneBookTester {
   public static void main(String[] args) {
      PhoneBook pb = new PhoneBook();
      pb.add("Scott", "0809876543");
      pb.add("Bill", "0909988776");
      pb.add("James", "0909090909");
      pb.add("Scott", "031234567#001");
      pb.remove("Scott", "0809876543");
      System.out.println("Scott");
      for (String num : pb.get("Scott")) {
         System.out.println("  " + num);
      }
   }
}
```

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。

解答群

ア　add(name)　　イ　add(phoneNumber)

ウ　put(name, numbers)　　エ　put(name, phoneNumber)

オ　put(numbers, name)　　カ　put(phoneNumber, name)

キ　remove(name)　　ク　remove(phoneNumber)

設問2　クラス PhoneBook に図2のメソッドを追加し，機能を拡張した。このメソッドに関する記述として正しい答えを，解答群の中から選べ。

```
public Set<String> methodX(String var) {
   Set<String> ret = new HashSet<String>();
   for (String key : book.keySet()) {
      if (key.startsWith(var)) {
         ret.add(key);
      }
   }
   return ret;
}
```

図2　クラス PhoneBook に追加したメソッド

解答群

ア　電話帳に登録されている電話番号のうち，引数 var で与えられた文字列で始まるすべての電話番号の集合を返す。

イ　電話帳に登録されている名前のうち，引数 var で与えられた文字列で始まるすべての名前の集合を返す。

ウ　引数 var で与えられた文字列で始まる電話番号に対応付けられたすべての名前の集合を返す。

エ　引数 var で与えられた文字列で始まる名前に対応付けられたすべての電話番号の集合を返す。

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語のマスクパターンを生成するプログラムである。

(1) プログラムと生成するマスクパターンの対応を表1に示す。

表1　プログラムと生成するマスクパターンの対応

プログラム名	マスクパターンの生成
PTN1	最上位ビットから与えられたビット番号までのビットを1，それ以外を0に設定
PTN2	最上位ビットから与えられたビット番号までのビットを0，それ以外を1に設定
PTN3	与えられたビット番号のビットを1，それ以外を0に設定

(2) プログラムは，ビット番号（0～15）をGR1に設定して呼び出され，生成したマスクパターンをGR2に格納して主プログラムに戻る。

(3) プログラムから戻るとき，汎用レジスタGR1の内容は元に戻す。

〔プログラム1〕

```
PTN1    START
        PUSH  0,GR1
        LAD   GR2,15
        SUBL  GR2,GR1
        LD    GR1,GR2
        LD    GR2,=[   a   ]
        SRA   GR2,0,GR1
        POP   GR1
        RET
        END
```

〔プログラム2〕

```
PTN2    START
        PUSH  0,GR1
        LAD   GR2,16
        SUBL  GR2,GR1
        LD    GR1,GR2
        LD    GR2,=[   b   ]
        SRL   GR2,0,GR1
        POP   GR1
        RET
        END
```

〔プログラム3〕

```
PTN3    START
        LD    GR2,=[   c   ]
        SLL   GR2,0,GR1
        RET
        END
```

アセンブラ

設問1　プログラム1～3中の［　　］に入れる正しい答えを，解答群の中から選べ。

解答群

ア　#0001　　イ　#0FFF

ウ　#8000　　エ　#FFF0

オ　#FFFF

設問2　PTN1，PTN2を使用して，表2に示すマスクパターンを生成するPTN4を作成した。プログラム4中の［　　］に入れる正しい答えを，解答群の中から選べ。

表2　作成したプログラムと生成するマスクパターンの対応

プログラム名	マスクパターンの生成
PTN4	与えられたビット番号B1からB2（15＞B1≧B2＞0）までのビットを0，それ以外を1に設定

プログラムは，B1をGR1に，B2をGR2に設定して呼び出され，生成したマスクパターンをGR2に格納して主プログラムに戻る。プログラムから戻るとき，汎用レジスタGR1の内容は元に戻す。

〔プログラム4〕

```
PTN4    START
        ST      GR1,WRK
        LD      GR1,GR2
        CALL    PTN1
        PUSH    0,GR2           ; PTN1 の結果を一時保存
        LD      GR1,WRK
        LAD     GR1,1,GR1       ; ビット番号の調整
        CALL    PTN2
        POP     GR1             ; 保存していた PTN1 の結果を GR1 へ
        [      d      ]
        LD      GR1,WRK
        RET
WRK     DS      1
        END
```

解答群

ア　AND　GR2,GR1　　イ　OR　GR2,GR1

ウ　SLL　GR2,0,GR1　　エ　SRA　GR2,0,GR1

オ　SRL　GR2,0,GR1　　カ　XOR　GR2,GR1

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〕

与えられた平文(ひらぶん)を，換字(かえじ)表を用いて暗号文に変換する関数 encrypt_text である。

(1) 関数 encrypt_text の引数は，次のとおりである。ただし，ファイル名に誤りはないものとする。

in_filename	平文が格納されているファイル名
out_filename	暗号文を格納するファイル名
ctbl	換字表

(2) 平文に含まれるものは，次の92種類の文字である。

① 英字 A～Z，a～z

② 数字 0～9

③ 記号 ! " # % & ' ( ) * + , - . / : ; < = > ? [\ ] ^ _ { | } ~

④ 空白文字

(3) 換字表は，平文に含まれる92種類の文字を格納した4行23列の2次元文字型配列である。換字表のすべての要素には，異なる文字が格納される。

(4) 換字による暗号化は，次のとおりに行う。ただし，換字表の最下行（行3）の下には最上行（行0）が，最右列（列22）の右には最左列（列0）があるものとして処理を行う。

① 平文の先頭から2文字ずつを取り出し，②～⑥の処理を行う。平文の文字数が奇数の場合は，最後の1文字について，後ろに空白文字1文字を追加して2文字にして処理を行う。

② それぞれの文字について，換字表の中の位置（行と列）を求める。

③ 2文字が同じ位置にある（同一の文字である）場合

それぞれの文字を，換字表の右下（1行下で1列右）にある文字に置き換える。換字の例を図1中の③に示す。

④　2文字が同じ行にある場合

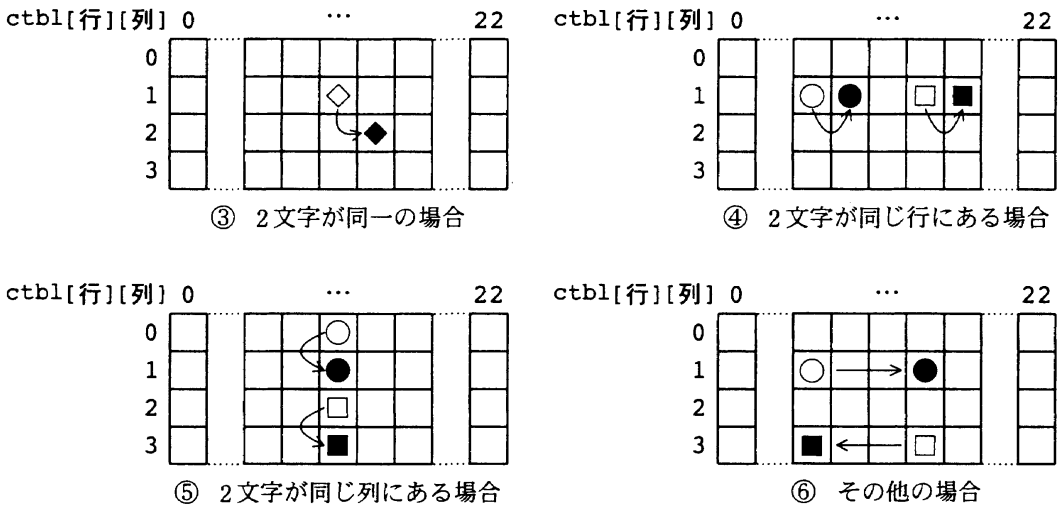
それぞれの文字を，換字表の右隣（同じ行の1列右）にある文字に置き換える。換字の例を図1中の④に示す。

⑤　2文字が同じ列にある場合

それぞれの文字を，換字表の直下（1行下の同じ列）にある文字に置き換える。換字の例を図1中の⑤に示す。

⑥　その他の場合

それぞれの文字を，換字表の同じ行で，他方の文字と同じ列にある文字に置き換える。換字の例を図1中の⑥に示す。



（凡例）

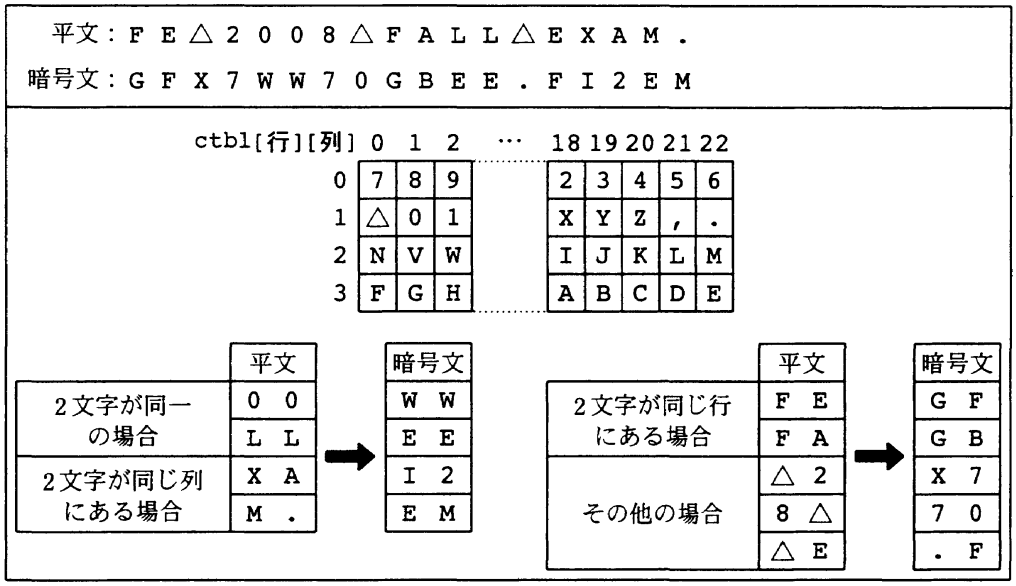
◇：平文の1，2文字目　　◆：1，2文字目の換字の結果

○：平文の1文字目　　●：1文字目の換字の結果

□：平文の2文字目　　■：2文字目の換字の結果

図1　換字の例

(5) 換字表を用いた暗号化の例を図2に示す。



注　“△”は空白文字を表す。

図2　換字表を用いた暗号化の例

〔プログラム〕

（行番号）

```
 1  #include <stdio.h>
 2
 3  #define ROWS    4     /* 換字表の行数 */
 4  #define COLS    23    /* 換字表の列数 */
 5
 6  void encrypt_text(const char *, const char *,
 7                    const char[ROWS][COLS]);
 8
 9  void encrypt_text(const char *in_filename,
10                    const char *out_filename,
11                    const char ctbl[ROWS][COLS]){
12     FILE *ifp, *ofp;
13     char ch[2];
14     int col[2], row[2], flg, i, sts;
15
16     ifp = fopen(in_filename, "r");
17     ofp = fopen(out_filename, "w");
```

```
18      do{
19          sts = fgetc(ifp);
20          if(sts != EOF){
21              ch[0] = sts;
22              sts = fgetc(ifp);
23              if( [ a ] ){    /* 文字数が奇数の場合 */
24                  ch[1] = ' ';
25              }else{
26                  ch[1] = sts;
27              }
28              for(i = 0; i < 2; i++){
29                  flg = 0;
30                  for(row[i] = 0; row[i] < ROWS; row[i]++){
31                      for(col[i] = 0; col[i] < COLS; col[i]++){
32                          if(ch[i] == ctbl[row[i]][col[i]]){
33                              flg = 1;
34                              break;
35                          }
36                      }
37                      if(flg != 0) break;
38                  }
39              }
40              if( [ b ] ){
41                  if( [ c ] ){      /* 2 文字が同一の場合 */
42                      ch[0] = ch[1] = [ d ];
43                  }else{              /* 2 文字が同じ行にある場合 */
44                      ch[0] = ctbl[row[0]][(col[0]+1) % COLS];
45                      ch[1] = ctbl[row[1]][(col[1]+1) % COLS];
46                  }
47              }else if( [ c ] ){  /* 2 文字が同じ列にある場合 */
48                  ch[0] = ctbl[(row[0]+1) % ROWS][col[0]];
49                  ch[1] = ctbl[(row[1]+1) % ROWS][col[1]];
50              }else{
51                  ch[0] = ctbl[row[0]][col[1]];
52                  ch[1] = ctbl[row[1]][col[0]];
53              }
54              fputc(ch[0], ofp);
55              fputc(ch[1], ofp);
56          }
57      }while(sts != EOF);
58      fclose(ifp);
59      fclose(ofp);
60  }
```

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　`ch[0] != ' '`　　イ　`ch[0] != ch[1]`　　ウ　`ch[0] == ' '`

エ　`ch[0] == ch[1]`　　オ　`sts != EOF`　　カ　`sts == EOF`

b，cに関する解答群

ア　`ch[0] != ch[1]`　　イ　`ch[0] == ch[1]`

ウ　`col[0] != col[1]`　　エ　`col[0] == col[1]`

オ　`row[0] != row[1]`　　カ　`row[0] == row[1]`

dに関する解答群

ア　`ctbl[(col[0]+1) % ROWS][(row[0]+1) % COLS]`

イ　`ctbl[col[0]+1][row[0]+1]`

ウ　`ctbl[col[0]][row[0]]`

エ　`ctbl[(row[0]+1) % ROWS][(col[0]+1) % COLS]`

オ　`ctbl[row[0]+1][col[0]+1]`

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。

(1) 平文に含まれる文字として改行文字を追加し，換字による暗号化を次のとおりに変更する。

（新しい方法）

平文から改行文字を除いた文を従来の換字規則で暗号化し，平文と同じ位置に改行文字を挿入したものを暗号文とする。

(2) 図2の換字表を用いた新しい方法での暗号化の例を図3に示す。

```
平文：F E △ 2 0 0 8 ↓
      ↓
      △ F A L L △ E X A M . ↓

暗号文：G F X 7 W W 7 ↓
      ↓
      0 G B E E . F I 2 E M ↓
```

注　“△”は空白文字を，“↓”は改行文字を表す。

図3　新しい方法での暗号化の例

(3) これに対応するために，プログラムを表のとおりに変更する。

表　プログラムの変更内容

処置	変更内容
行番号14と15の間に追加	`int cnt2;`

行番号19を変更	`sts = fgetc(ifp);`
	`while((char)sts == '\n'){`
	`  fputc('\n', ofp);`
	`  sts = fgetc(ifp);`
	`}`

行番号22を変更	`cnt2 = 0;`
	`sts = fgetc(ifp);`
	`while((char)sts == '\n'){`
	`  cnt2++;`
	`  sts = fgetc(ifp);`
	`}`

[　e　]の間に追加	`for( ; cnt2 > 0; cnt2--){`
	`  fputc('\n', ofp);`
	`}`

解答群

ア　行番号53と54　　イ　行番号54と55

ウ　行番号55と56　　エ　行番号56と57

オ　行番号57と58

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〕

論理式評価プログラム**EVAL-EXP**の動作検証のため，論理式を構成するすべてのパターンの文字列を生成するプログラム**GEN-EXP**である。

(1) **GEN-EXP**は，1文字から7文字までの論理式文字列を自動生成して**EVAL-EXP**を呼び出し，図1のような結果を印字する。検証はこの結果リストを使って目視で行う。見出しは印刷済とする。

```
評価結果：   論理式
T:           T
F:           F
E:           (
E:           &
 ⋮           ⋮
F:           T&T&F
 ⋮           ⋮
F:           T&T&(F)
 ⋮           ⋮
```

図1　結果の例

(2) 論理式評価プログラムの呼出しは，次のように行う。

```
CALL "EVAL-EXP" USING パラメタ1　パラメタ2
```

ここで，**パラメタ1**は8けたで，空白で終了する1文字から7文字の論理式を指定する。**パラメタ2**は評価結果であり，真のときは**T**，偽のときは**F**，構文エラーがあるときは**E**が返る。**パラメタ1**は**EVAL-EXP**の実行によって変更されることはない。

(3) **EVAL-EXP**で扱う論理式は，次の仕様とする。

① 論理式は，論理項目**T**，**F**，二つの論理式を論理演算子**&**，**|**でつないだもの，及び論理式を括弧**(**，**)**でくくったもので構成される。ここで，**T**は真値，**F**は偽値，**&**は論理積，**|**は論理和を表す。

② 論理式の評価の順序は，括弧でくくられた内側が先に評価され，括弧でくくられていないか又は括弧が同じ順位の場合は**&**が**|**よりも優先順位が高い。同じ優先順位が並ぶ場合は左から右に評価される。

(4) 論理式の自動生成は，数字１～６からなる長さ１～７の数字列を，作業領域NUM-TBLに生成し，数字1～6に6種類の文字T，F，(，&，|，)を対応させて変換する（図2）。

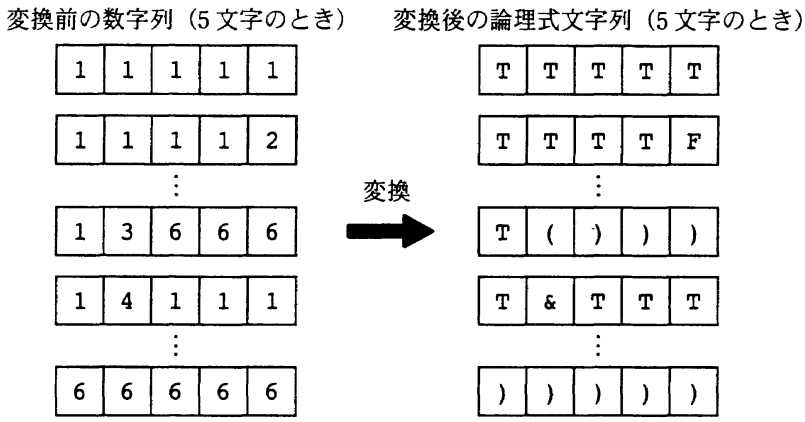


図2　作業領域NUM-TBLに生成された数字列と対応する論理式文字列

〔プログラムGEN-EXP〕

（行番号）

```
 1  DATA DIVISION.
 2  FILE SECTION.
 3  FD  EXP-F.
 4  01  EXP-REC                PIC X(08).
 5  FD  PRINT-F.
 6  01  P-REC                  PIC X(13).
 7  WORKING-STORAGE SECTION.
 8  01  CONSTANT-CHARS.
 9      05  CONS-VALUE         PIC X(06) VALUE "TF(&|)".
10      05  FILLER             REDEFINES CONS-VALUE.
11          10  CONS-TBL       PIC X OCCURS 6.
12  01  NUM-BUF.
13      05  NUM-TBL            PIC 9 OCCURS 10.
14  01  EXIT-SW                PIC 9.
15  01  EXP-L                  PIC 99.
16  01  RESULT-SW              PIC X.
17  01  WK-I                   PIC 99.
18  01  WK-TEMP                PIC 99.
19  PROCEDURE DIVISION.
20  MAIN-CTL.
21      OPEN OUTPUT EXP-F.
22      PERFORM GENERATE-EXP VARYING EXP-L FROM 1 BY 1
23                                          UNTIL  EXP-L > 7.
24      CLOSE EXP-F.
```

```
*
     OPEN INPUT  EXP-F.
     OPEN OUTPUT PRINT-F.
     PERFORM EVAL-EXP-PRINT.
     CLOSE EXP-F PRINT-F.
*
     STOP RUN.
 GENERATE-EXP.
     MOVE SPACE   TO EXP-REC.
*    作業領域初期化
     MOVE ALL "T" TO EXP-REC(1:EXP-L).
     PERFORM VARYING WK-I FROM 1 BY 1 UNTIL WK-I > EXP-L
       [ a ]
     END-PERFORM.
     MOVE EXP-L TO WK-I.
*    数字列の生成と論理式への変換
     PERFORM [ b ]
       WRITE EXP-REC
       MOVE EXP-L TO WK-I
       MOVE 0     TO EXIT-SW
       PERFORM UNTIL WK-I < 1 OR EXIT-SW = 1
         [ c ]
*        けた上がりのチェック
         IF NUM-TBL(WK-I) > 6 THEN
            MOVE 1 TO NUM-TBL(WK-I)
            MOVE CONS-TBL(1) TO EXP-REC(WK-I:1)
            SUBTRACT 1 FROM WK-I
         ELSE
            MOVE NUM-TBL(WK-I) TO WK-TEMP
            MOVE CONS-TBL(WK-TEMP) TO EXP-REC(WK-I:1)
            MOVE 1 TO EXIT-SW
         END-IF
       END-PERFORM
     END-PERFORM.
*
 EVAL-EXP-PRINT.
     MOVE 0 TO EXIT-SW.
     PERFORM UNTIL EXIT-SW = 1
       READ EXP-F AT END MOVE 1 TO EXIT-SW
              NOT AT END
                 CALL "EVAL-EXP" USING EXP-REC RESULT-SW
                 STRING RESULT-SW ":    " EXP-REC
                        DELIMITED BY SIZE INTO P-REC
                 WRITE P-REC AFTER 1
       END-READ
     END-PERFORM.
```

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。

解答群

ア　ADD 1 TO NUM-TBL(WK-I)　　イ　ADD 1 TO WK-I

ウ　MOVE 1 TO NUM-TBL(WK-I)　　エ　SUBTRACT 1 FROM WK-I

オ　UNTIL EXIT-SW = 1　　カ　UNTIL WK-I < 1

キ　UNTIL WK-I > EXP-L

設問2　GEN-EXP では，論理式を構成するすべてのパターンの文字列を生成しているので，評価結果のほとんどが構文エラーであった。そこで，EVAL-EXP での論理式評価が効率よく実行できるよう，GEN-EXP で論理式生成後に論理式の構文チェックを行い，構文エラーでないものだけを検査対象とするように変更することとなった。論理式の構文チェックは次の①，②に従う。

①　論理式で許される文字 T，F，(，&，|，) の記述の順序の可否は表1のとおりとする。

表1　記述の順序

左側＼右側	T，F又は (	&，｜又は)
T，F又は)	－	○
&，｜又は (	○	－

注　○：許される，－：許されない

②　論理式の先頭文字には &，|，) は許されず，論理式の末尾文字には &，|，(は許されない。また，左右の括弧は対になっていなければならない。

このためにプログラムを次のとおりに変更する。表2中の［　　］に入れる正しい答えを，解答群の中から選べ。

表2　プログラムの変更内容

処置	変更内容
行番号18と19の間に追加	01　CHECK-ERROR　　PIC 9. 01　NEST-LEVEL　　PIC S99. 01　WK-R　　PIC 99. 01　WK-L　　PIC 99.
行番号42を変更	PERFORM CHECK-SYNTAX IF CHECK-ERROR = 0 THEN 　WRITE EXP-REC END-IF

行番号70の次に追加	CHECK-SYNTAX.
	　MOVE 0 TO CHECK-ERROR.
	　MOVE 0 TO NEST-LEVEL.
	　PERFORM VARYING WK-R FROM 1 BY 1
	　　UNTIL WK-R > EXP-L OR CHECK-ERROR NOT = 0
	*　先頭文字チェック
	　IF WK-R = 1 THEN
	　　IF EXP-REC(WK-R:1) = "&" OR "|" OR ")" THEN
	　　MOVE 1 TO CHECK-ERROR
	　　END-IF
	　ELSE
	　　COMPUTE WK-L = WK-R - 1
	*　　記述順序のチェック
	　　IF EXP-REC(WK-L:1) = "(" OR "&" OR "|" THEN
	　　IF EXP-REC(WK-R:1) = "&" OR "|" OR ")" THEN
	　　MOVE 1 TO CHECK-ERROR
	　　END-IF
	　　ELSE
	　　IF EXP-REC(WK-R:1) = [　d　] THEN
	　　MOVE 1 TO CHECK-ERROR
	　　END-IF
	　　END-IF
	　END-IF
	　EVALUATE EXP-REC(WK-R:1)
	　　WHEN "(" [　e　]
	　　WHEN ")" SUBTRACT 1 FROM NEST-LEVEL
	　　IF NEST-LEVEL < 0 THEN
	　　MOVE 1 TO CHECK-ERROR
	　　END-IF
	　END-EVALUATE
	　END-PERFORM.
	*　最右端のチェック
	　IF CHECK-ERROR = 0 AND (EXP-REC(EXP-L:1) = "&" OR
	　　"|" OR "(") THEN
	　　MOVE 1 TO CHECK-ERROR
	　END-IF.
	*　括弧の対応チェック
	　IF CHECK-ERROR = 0 AND NEST-LEVEL NOT = 0 THEN
	　　MOVE 1 TO CHECK-ERROR
	　END-IF.

解答群

ア "&" OR "|" OR ")"　　イ "T" OR "F" OR "("

ウ "T" OR "F" OR ")"　　エ ADD 1 TO NEST-LEVEL

オ SUBTRACT 1 FROM NEST-LEVEL

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1，2に答えよ。

（Javaプログラムで使用するAPIの説明は，この冊子の末尾を参照してください。）

〔プログラムの説明〕

文字列照合用パターン（以下，パターンという）と，与えられた文字列を照合するプログラムである。パターン及び与えられた文字列は0個以上の文字からなる文字列である。パターンを構成する文字には，照合するときに特殊な意味をもつ文字（以下，メタ文字という）及びメタ文字以外の文字（以下，通常文字という）がある。表1にメタ文字とその意味を示す。

表1　メタ文字とその意味

メタ文字	意味
^	文字列の先頭に一致
.	任意の1文字に一致
$	文字列の末尾に一致

通常文字は，その文字と一致する。例えば，通常文字 'a' は1文字のaに一致する。

与えられた文字列の全部又は一部にパターン全体が一致した場合，一致したものと判定する。パターンの文字数が0のとき，すべての文字列に一致する。表2にパターンとそのパターンに一致する文字列及び一致しない文字列の例を示す。

表2　パターンとそのパターンに一致する文字列及び一致しない文字列の例

パターン	一致する文字列の例	一致しない文字列の例
""	"", "home"	なし
"home"	"home", "tachometer"	"how", "hope", "me"
"^ho.e"	"home", "hope", "hotel"	"hot", "hook", "shore"
"ho..$"	"home", "hope", "shore"	"hop", "hotel", "hobby"

プログラムは，与えられたパターンを1文字ずつ解析し，照合処理がしやすい形式に変換する。この解析及び変換処理をコンパイルと呼び，変換された照合処理の単位をパターン要素と呼ぶ。メタ文字は，そのメタ文字の意味を表すパターン要素に変換され，通常文字は，その文字を表すパターン要素に変換される。パターン全体は，

パターン要素のリストで表される。例えば，パターン"^ho.e"は，'^'，'h'，'o'，'.'及び'e'に分けられ，それぞれが次のパターン要素に変換される。

① 文字列の先頭を表すパターン要素

② 'h'を表すパターン要素

③ 'o'を表すパターン要素

④ 任意の1文字を表すパターン要素

⑤ 'e'を表すパターン要素

次に，パターンと与えられた文字列との照合を，パターン要素のリストを評価しながら行う。リスト中のパターン要素（前の例では①～⑤）と，与えられた文字列の先頭の文字から順に照合し，連続してすべてのパターン要素が一致したとき，パターンは与えられた文字列に一致すると判定する。

インタフェース PatternElement は，パターン要素を表す。メソッド matches は，このパターン要素が，引数で与えられた文字列 str の位置 index から一致するかどうかを調べ，一致すれば true を，それ以外は false を返す。ただし，index の値は0以上であるものとする。メソッド length は，このパターン要素に一致する文字列の長さを返す。

インタフェース PatternElement を実装した各クラスを次に示す。

(1) クラス OneChar は，通常文字1文字を表すパターン要素である。

(2) クラス AnyChar は，任意の1文字を表すパターン要素である。

(3) クラス BeginningOfString 及び EndOfString は，それぞれ文字列の先頭及び文字列の末尾を表すパターン要素である。

クラス Pattern は，コンストラクタで与えられたパターンをパターン要素のリストに変換する。メソッド matches は，与えられた文字列とパターン要素のリストを照合し，一致すれば true を，それ以外は false を返す。メソッド main はクラス Pattern のテストを行う。メソッド main を実行すると，図1のような出力結果が得られる。

```
"ho.e$" matches "home".
"ho.e$" matches "shore".
"ho.e$" doesn't match "hotel".
```

図1　出力結果

なお，このプログラムでは，1文字が char で表現できるものとし，Unicode の補助文字は考えないものとする。

〔プログラム1〕

```
public interface PatternElement {
   public boolean matches(String str, int index);
   public int length();
}
```

〔プログラム2〕

```
class OneChar implements PatternElement {
   private final char ch;
   OneChar(char ch) { this.ch = ch; }
   // 与えられた文字列 str の位置 index の文字とこのパターン要素が
   // 表す 1 文字が一致すれば true を，それ以外は false を返す。
   public boolean matches(String str, int index) {
      return str.length() > index && [   a   ];
   }
   public int length() { return 1; }
}
```

〔プログラム3〕

```
class AnyChar implements PatternElement {
   // 与えられた文字列 str に，位置 index から文字が一つ以上あれば true を，
   // それ以外は false を返す。
   public boolean matches(String str, int index) {
      return [   b   ];
   }
   public int length() { return 1; }
}
```

〔プログラム4〕

```
class BeginningOfString implements PatternElement {
   public boolean matches(String str, int index) {
      return index == 0;
   }
   public int length() { return 0; }
}
```

〔プログラム5〕

```
class EndOfString implements PatternElement {
   public boolean matches(String str, int index) {
      return index == str.length();
   }
   public int length() { return 0; }
}
```

〔プログラム6〕

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

public class Pattern {
   private List<PatternElement> expr;
   public Pattern(String pattern) {
      expr = compile(pattern);
   }

   public boolean matches(String str) {
      for (int i = 0; i <= str.length(); i++) {
         if (matches(str, i))
            return true;
      }
      return false;
   }

   private boolean matches(String str, int index) {
      for (PatternElement node : expr) {
         if (!node.matches(str, index))
            return false;
         index += node.length();
      }
      return true;
   }

   private List<PatternElement> compile(String pattern) {
      List<PatternElement> list = new ArrayList<PatternElement>();
      for (int i = 0; i < pattern.length(); i++) {
         char c = pattern.charAt(i);
         PatternElement node = null;
         if (c == '.') {
            node = new [ c ];
         } else if (c == '^') {
            node = new BeginningOfString();
         } else if (c == '$') {
            node = new EndOfString();
         } else {
            node = new [ d ];
         }
         list.add(node);
      }
      return list;
   }
```

α

Java

```
    public static void main(String[] args) {
       String[] data = { "home", "shore", "hotel" };
       Pattern pattern = new Pattern("ho.e$");
       for (String str : data) {
          String result;
          if ( [ e ] ) {
             result = "matches";
          } else {
             result = "doesn't match";
          }
          System.out.printf("\"ho.e$\" %s \"%s\".%n",
                            result, str);
       }
    }
 }
```

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　str.charAt(index) != ch　　イ　str.charAt(index) < ch

ウ　str.charAt(index) == ch　　エ　str.charAt(index) > ch

bに関する解答群

ア　str.length() < index　　イ　str.length() <= index

ウ　str.length() > index　　エ　str.length() >= index

c, dに関する解答群

ア　AnyChar('.')　　イ　AnyChar()　　ウ　AnyChar(c)

エ　AnyChar(i)　　オ　OneChar('.')　　カ　OneChar()

キ　OneChar(c)　　ク　OneChar(i)

eに関する解答群

ア　!pattern.equals(str)　　イ　!pattern.matches(str)

ウ　pattern.equals(str)　　エ　pattern.matches(str)

設問2　プログラム6の α の行を図2のとおりに変更したとき，プログラムの動作はどうなるか。適切な記述を，解答群の中から選べ。

```
} else if (c == '^' && i == 0) {
```

図2　プログラム6の α の行の変更内容

解答群

ア　文字'^'は，常に通常文字として扱われる。

イ　文字'^'は，引数patternの先頭にあるときだけメタ文字として扱われ，それ以外のときは，通常文字として扱われる。

ウ　文字'^'は，引数patternの先頭にあるときだけメタ文字として扱われ，それ以外のときは，無視される。

エ　文字'^'は，引数patternの先頭にあるときだけメタ文字として扱われ，それ以外のときは，例外が発生する。

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1，2に答えよ。

〔プログラムの説明〕

アンケートに対するn人分の回答を集計し，集計結果をグラフで出力する副プログラムSUMMARYである。

アンケートの質問は16項目からなり，“はい”又は“いいえ”で答える。副プログラムSUMMARYは，項目ごとに“はい”と答えた人数を集計し，棒グラフで出力する。このグラフの表示形式を図1に示す。横軸は項目番号，縦軸は“はい”と答えた人数であり，一つの“*”が1人を表す。□は空白文字を表す。

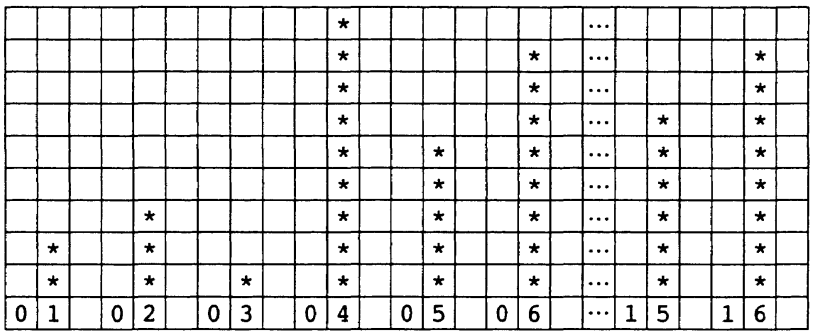


図1　グラフの表示形式

(1) 1人分の回答は図2のように1語に格納され，左端のビットから順に項目番号1～16に対応している。“はい”が1，“いいえ”が0として格納されている。

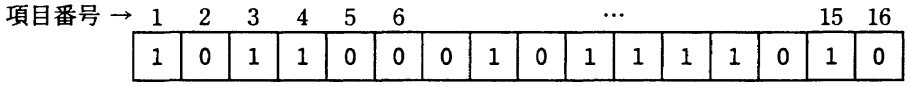


図2　1人分の回答の格納形式

(2) n人分の回答が，図3のように連続するn語の領域に格納され，その先頭アドレスがGR1に設定されて主プログラムから渡される。

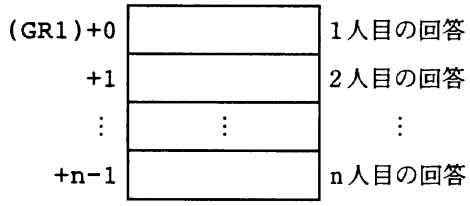


図3　主プログラムから渡されるn人分の回答

(3) 値 n（1≦n≦100）はGR0に設定されて，主プログラムから渡される。

(4) 副プログラムSUMMARYは，まず，項目ごとに“はい”と答えた人数を数え，図4に示す16語の領域COUNTRに格納する。

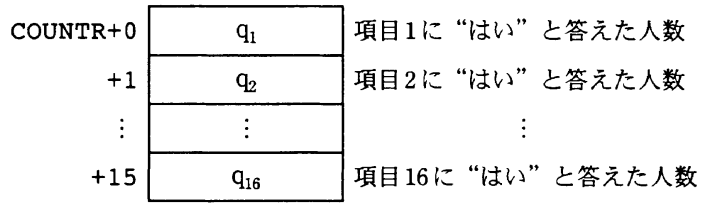


図4　集計結果を格納するカウンタ領域

次に，この集計結果を図1で示した棒グラフで出力する。グラフは，上から順に1行ずつ出力する（出力行数は，q_1～q_{16}の最大値に1を加えたものとなる）。

(5) 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。

〔プログラム〕

（行番号）

```
 1  SUMMARY START
 2          RPUSH
 3  ; カウンタ領域と出力バッファを初期化
 4          LD     GR2,=0
 5          LD     GR3,=' '
 6          LD     GR4,=0              ; ループカウンタ
 7          LAD    GR5,PBUF            ; 出力バッファのポインタ
 8  LOOP1   ST     GR2,COUNTR,GR4      ; カウンタ領域を初期化
 9          ST     GR3,0,GR5           ; 出力バッファを空白で初期化
10          ST     GR3,1,GR5
11          ST     GR3,2,GR5
12          [      a      ]            ; 出力バッファのポインタを更新
13          LAD    GR4,1,GR4           ; ループカウンタを更新
14          CPA    GR4,=16
15          JMI    LOOP1
16  ; 項目ごとに“はい”と答えた人数を集計
17          LD     GR5,=0              ; GR5:“はい”の人数の最大値
18  LOOP2   LD     GR4,=0              ; カウンタ領域のポインタ
19          LD     GR2,0,GR1           ; GR2 ← 1 人分の回答
20  LOOP3   SLL    GR2,1               ; 回答は“はい”？
21          [      b      ]
22          JZE    NEXTW               ; 残りの項目はすべて“いいえ”
23          JUMP   OFF
```

```
24  ON      LD    GR3,COUNTR,GR4    ; カウンタに 1 を加算
25          ADDA  GR3,=1
26          ST    GR3,COUNTR,GR4
27          CPA   GR3,GR5           ; 最大値と比較
28          JPL   CHANGE
29          JUMP  OFF
30  CHANGE  LD    GR5,GR3           ; 最大値を入替え
31  OFF     LAD   GR4,1,GR4         ; 次の項目
32          JUMP  LOOP3
33  NEXTW   [        c        ]
34          SUBA  GR0,=1            ; 全回答処理済？
35          JPL   LOOP2
36  ; 集計結果を棒グラフで出力
37          LD    GR5,GR5
38          JZE   FIN
39          LD    GR0,='*'
40  LOOP4   LD    GR3,=1            ; 出力バッファのポインタ
41          LD    GR4,=0            ; カウンタ領域のポインタ
42  LOOP5   CPA   GR5,COUNTR,GR4    ; “はい”の人数と比較
43          JNZ   NOTSET
44          ST    GR0,PBUF,GR3      ; 出力バッファに“*”を設定
45  NOTSET  LAD   GR3,3,GR3         ; 出力バッファのポインタを更新
46          LAD   GR4,1,GR4         ; カウンタ領域のポインタを更新
47          CPA   GR4,=16
48          JMI   LOOP5
49          OUT   PBUF,PLEN
50          SUBA  GR5,=1
51          [        d        ]
52  FIN     OUT   FOOTER,PLEN
53          RPOP
54          RET
55  COUNTR  DS    16                ; カウンタ領域
56  PLEN    DC    48
57  PBUF    DS    48                ; 出力バッファ
58  FOOTER  DC    '01 02 03 04 05 06 07 08 09 10 11 12 13 '
59          DC    '14 15 16 '
60          END
```

アセンブラ

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　LAD　GR5,1,GR5　　イ　LAD　GR5,2,GR5

ウ　LAD　GR5,3,GR5　　エ　LAD　GR5,4,GR5

bに関する解答群

ア	JMI　ON	イ	JNZ　ON
ウ	JOV　ON	エ	JPL　ON

cに関する解答群

ア	LAD　GR1,-1,GR1	イ	LAD　GR1,1,GR1
ウ	LAD　GR2,-1,GR2	エ	LAD　GR2,1,GR2

dに関する解答群

ア	JPL　LOOP4	イ	JPL　LOOP5
ウ	JZE　LOOP4	エ	JZE　LOOP5

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。

主プログラムから渡されるnの値は3で，3人分の回答は図5のとおりとする。

1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

図5　3人分の回答

このとき，行番号20のSLL命令は［　e　］回実行され，行番号44のST命令は［　f　］回実行される。

解答群

ア	4	イ	6	ウ	11	エ	16
オ	17	カ	20	キ	48		

■ Javaプログラムで使用するAPIの説明

java.util

public interface Map<K, V>

型Kのキーに型Vの値を対応付けて保持するインタフェースを提供する。各キーは，一つの値としか対応付けられない。

メソッド

---

public V get(Object key)

指定されたキーに対応付けられた値を返す。

引数： key — キー

戻り値：指定されたキーに対応付けられた型Vの値
このキーと値の対応付けがなければ null

---

public Set<K> keySet()

登録されているキーの集合を返す。

戻り値：登録されているキーの集合

---

public V put(K key, V value)

指定されたキーに指定された値を対応付けて登録する。このキーが既にほかの値と対応付けられていれば，その値は指定された値に置き換えられる。

引数： key — キー
value — 値

戻り値：指定されたキーに対応付けられていた型Vの古い値
このキーと値の対応付けがなければ null

---

public V remove(Object key)

指定されたキーの対応付けが登録されていれば，削除する。

引数： key — キー

戻り値：指定されたキーに対応付けられていた型Vの値
このキーと値の対応付けがなければ null

java.util

public class HashMap<K, V>

インタフェース Map のハッシュを用いた実装である。キー及び値は，null でもよい。

コンストラクタ

public HashMap()
空の HashMap を作る。

メソッド

public V get(Object key)
インタフェース Map のメソッド get と同じ

public Set<K> keySet()
インタフェース Map のメソッド keySet と同じ

public V put(K key, V value)
インタフェース Map のメソッド put と同じ

public V remove(Object key)
インタフェース Map のメソッド remove と同じ

java.util

public interface Set<E>

型 E の要素を集合（セット）として管理するインタフェースを提供する。
インタフェース Collection を継承する。

メソッド

public boolean add(E e)
指定された要素が集合に含まれていなければ，集合に追加する。
引数： e — 集合に追加される要素
戻り値：指定された要素が集合に含まれていなければ true
それ以外は false

public boolean addAll(Collection<? extends E> c)
指定されたコレクションの要素のうち，この集合に含まれていないものすべてを集合に追加する。
戻り値：この呼出しの結果，この集合が変更されれば true
それ以外は false

public boolean isEmpty()
空集合か否かを判定する。
戻り値：空集合ならば true
それ以外は false

public boolean remove(Object o)
指定された要素が集合に含まれていれば，集合から削除する。
引数： o — 集合から削除する要素
戻り値：指定された要素が集合に含まれていれば true
それ以外は false

java.util

public class HashSet<E>

インタフェース Set のハッシュを用いた実装である。

コンストラクタ

`public HashSet()`
空の HashSet を作る。

メソッド

`public boolean add(E e)`
インタフェース Set のメソッド add と同じ

`public boolean addAll(Collection<? extends E> c)`
インタフェース Set のメソッド addAll と同じ

`public boolean isEmpty()`
インタフェース Set のメソッド isEmpty と同じ

`public boolean remove(Object o)`
インタフェース Set のメソッド remove と同じ

java.lang

public final class String

クラス String は文字列を表す。

メソッド

`public boolean matches(String regex)`
この文字列が指定された正規表現と一致するかどうかを判定する。
引数： regex — 正規表現
戻り値：この文字列が指定された正規表現と一致すれば true
それ以外は false

`public boolean startsWith(String prefix)`
この文字列が指定された接頭辞で始まるかどうかを判定する。
引数： prefix — 接頭辞
戻り値：この文字列が指定された接頭辞で始まれば true
それ以外は false

java.util

public interface List<E>

リスト（順序付けられたコレクション）のためのインタフェースを提供する。

メソッド

boolean add(E e)

指定された要素 e をリストの最後に追加する。

引数： e — リストに追加する要素

戻り値：true

java.util

public class ArrayList<E>

インタフェース List の配列による実装である。

コンストラクタ

public ArrayList

空のリストを作る。

メソッド

public boolean add(E e)

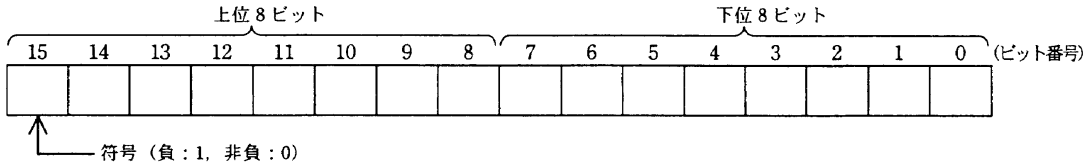
インタフェース List のメソッド add と同じ。

■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	SF	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 〕

7. 途中で退室する場合には，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室してください。

退室可能時間	13:40 ～ 15:20

8. **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。
9. 問題冊子の余白などは，適宜利用して構いません。
10. Java プログラムで使用する API の説明及び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。
11. 試験中，机上に置けるもの及び使用できるものは，次のものに限ります。
    なお，会場での貸出しは行っていません。
    受験票，黒鉛筆又はシャープペンシル，鉛筆削り，消しゴム，定規，時計（アラームなど時計以外の機能は使用不可），ハンカチ，ティッシュ
    これら以外は机上に置けません。使用もできません。
12. 試験終了後，この問題冊子は持ち帰ることができます。
13. 答案用紙は，いかなる場合でも，すべて提出してください。回収時に提出しない場合は，採点されません。
14. 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。