

平成 19 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。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は、試験開始の合図があってから始めてください。
4. 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

5. 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、〔問 6 と問 10 を選択した場合の例〕
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。
 - (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。

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	●				

〔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

アセンブリ

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。
<div style="display: flex; align-items: center;"> <div style="writing-mode: vertical-rl; font-size: 2em; margin-right: 10px;">処 理</div> <div style="flex-grow: 1;"> <ul style="list-style-type: none"> ・変数 ← 式 ・手続(引数, …) ▲ 条件式 ↓ 処理 ▲ 条件式 ↓ 処理 1 — 処理 2 ↓ ■ 条件式 ↓ 処理 ■ ■ 処理 ■ 条件式 ■ 変数: 初期値, 条件式, 増分 ↓ 処理 ■ </div> </div>	<ul style="list-style-type: none"> 変数に式の値を代入する。 手続を呼び出し、引数を受け渡す。 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。 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。 前判定繰返し処理を示す。 条件式が真の間、処理を繰返し実行する。 後判定繰返し処理を示す。 処理を実行し、条件式が真の間、処理を繰返し実行する。 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、処理を繰返す。また、繰返すごとに、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 ↑ ↓ 低
乗除演算	×, ÷, %	
加減演算	+, -	
関係演算	>, <, ≥, ≤, =, ≠	
論理積	and	
論理和	or	

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 浮動小数点表示法に関する次の記述を読んで、設問1, 2に答えよ。

$A \times 2^B$ の浮動小数点数（Aは仮数，Bは指数）を，図1に示す規格IEEE 754（IEC 60559）による単精度の浮動小数点形式では，次のように表現する。

(1) ビット番号31：符号部

A（仮数）の符号を表す。0のときは正を，1のときは負を示す。

(2) ビット番号30～23：指数部

B（指数）に127を加えた値を，2進数で表す。

(3) ビット番号22～0：仮数部

Bを調整することによって， $1 \leq |A| < 2$ となるように正規化を行い，更に整数部分の1を省略して，ビット番号22が小数第1位となるようにした2進数である。

ただし，数値が0の場合は，符号部，指数部，仮数部とも0とする。

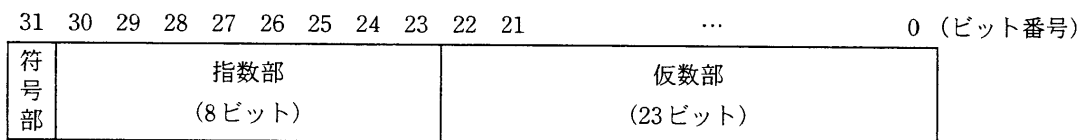


図1 規格IEEE 754による単精度の浮動小数点形式

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。

10進数の45.625を2進数で表すと， $(101101.101)_2$ となる。これを正規化した指数表現にすると， $(1.01101101)_2 \times 2^5$ となるので，IEEE 754の形式で表すと，図2に示すとおり符号部は $(0)_2$ ，指数部は指数の5に127を加えて $(10000100)_2$ となり，仮数部は整数部分を省略するので， $(011011010 \cdots 0)_2$ となる。

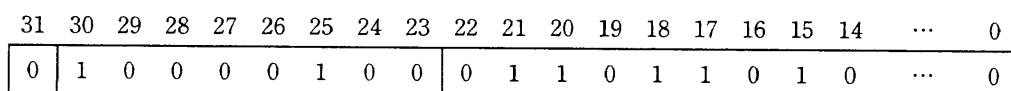


図2 IEEE 754の形式による45.625の表現

同様の手順で，10進数の -0.75 をIEEE 754の形式で表すと，符号部は()₂，指数部は()₂，仮数部の上位8けた(ビット番号22～15)は()₂となる。

解答群

- | | |
|------------|------------|
| ア 0 | イ 1 |
| ウ 01111110 | エ 01111111 |
| オ 10000000 | カ 11000000 |

設問2 IEEE 754の形式で表した，次の二つの浮動小数点数を加算した結果として正しい答えを，解答群の中から選べ。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	1	0	0	1	0	0	0	...	0

解答群

ア

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	0	0	1	0	0	1	0	...	0

イ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	0	0	1	1	1	0	0	...	0

ウ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	1	0	1	0	0	1	0	...	0

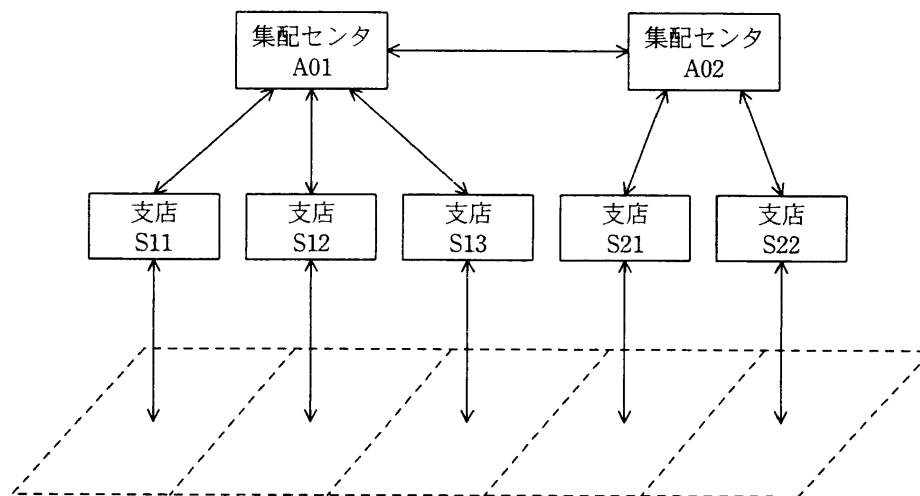
エ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	0
0	1	0	0	0	0	0	1	0	1	1	1	0	0	...	0

問2 関係データベースに関する次の記述を読んで、設問1～3に答えよ。

運送事業を営む X 社は、集配場所として、ある地域を網羅するように配置された支店と、それらの支店を管理する集配センタをもつ。支店が配達を受けもつ地域（以下、管轄地区という）は、重複せずに割り振られている。支店では、顧客から届け先に配達する品物（以下、配達品という）を受け付け、預り番号という X 社全体で一意的な番号を付与する。

支店は、配達品の届け先がその支店の受けもつ管轄地区にあれば配達を行い、そうでなければ、その支店を管理する集配センタに配達品を移送する。支店から支店への配達品の移送は必ず集配センタを経由する。集配センタでは、支店から移送された配達品を受け入れ、配達品の届け先を受けもつ支店がその集配センタの管理する支店であればその支店に、そうでなければ届け先を受けもつ支店を管理している集配センタに配達品を移送する。図1に配達品移送時の支店と集配センタの関係を示す。



注 破線の区画は、各支店の受けもつ管轄地区を示す。

図1 配達品移送時の支店と集配センタの関係

X 社では、図2に示す構造の関係データベースを用いて、配達品の状態を把握している。図中の下線はキー項目を表す。

配達地域表

管轄地区	支店コード
------	-------

支店表

支店コード	集配センタコード
-------	----------

配達品表

預り番号	受付日時	受付支店コード	依頼元管轄地区	依頼元住所	依頼元氏名	届け先管轄地区	届け先住所	届け先氏名
------	------	---------	---------	-------	-------	---------	-------	-------

移送履歴表

預り番号	通過日時	通過店コード	入出区分
------	------	--------	------

図2 関係データベースの構造

配達地域表：管轄地区ごとに、配達を受けもつ支店コード（Sで始まる3けた）をもつ。

支店表：支店コードごとに、支店を管理する集配センタコード（Aで始まる3けた）をもつ。

配達品表：預り番号ごとに、配達品の依頼元や届け先などの情報を保持する。

移送履歴表：配達品が集配場所（支店や集配センタ）を出入りするごとに、その日時と出入りした履歴を保持する。入出区分には配達品が集配場所に入るときには'I'を、出るときには'O'を格納する。

設問1 集配センタ A01 が管理している支店の数を求める SQL 文として正しい答えを、解答群の中から選べ。

解答群

- ア SELECT COUNT(支店コード) FROM 支店表
WHERE 集配センタコード NOT IN ('A01')
- イ SELECT COUNT(支店コード) FROM 支店表
WHERE 集配センタコード = 'A01'
- ウ SELECT COUNT(支店コード) FROM 支店表 WHERE 支店コード = 'A01'
- エ SELECT 支店コード FROM 支店表 WHERE 集配センタコード = 'A01'
- オ SELECT 集配センタコード FROM 支店表
WHERE 支店コード = 'A01' ORDER BY 集配センタコード

設問2 次の SQL 文は、支店 S11 でこれまでに受け付けた配達品のうち、その支店が直接配達を受けもった届け先の一覧を作成するものである。SQL 文中の に入れる正しい答えを、解答群の中から選べ。

```
SELECT 届け先管轄地区, 届け先住所, 届け先氏名 FROM  a
WHERE 受付支店コード = 'S11'  b
      届け先管轄地区  c (SELECT 管轄地区 FROM 配達地域表
      WHERE 支店コード = 'S11')
```

解答群

- | | | |
|----------|-------|---------|
| ア 移送履歴表 | イ 支店表 | ウ 配達地域表 |
| エ 配達品表 | オ AND | カ IN |
| キ NOT IN | ク OR | |

設問3 移送履歴表に対して次の SQL 文による問合せを行った。この問合せの結果として判明することを、解答群の中から選べ。

```
SELECT 通過日時, 通過店コード, 入出区分 FROM 移送履歴表
WHERE 預り番号 = '0000004' AND
      通過日時 = (SELECT MAX(通過日時) FROM 移送履歴表
      WHERE 預り番号 = '0000004')
```

解答群

- ア 預り番号が 0000004 である配達品が最も長い期間保管されていた支店
- イ 預り番号が 0000004 である配達品の依頼元
- ウ 預り番号が 0000004 である配達品のすべての移送履歴
- エ 預り番号が 0000004 である配達品の最新の集配場所と日時
- オ 預り番号が 0000004 である配達品を受け付けてから配達するまでに経過した日数と時間

問3 通信回線に関する次の記述を読んで、設問に答えよ。

A社は、本社と工場を結ぶ、X、Y、Zの3系統の独立した通信回線をもっている。

〔回線X、Y、Zの説明〕

- (1) X、Y、Zは、それぞれ運用開始年は異なるが、3系統とも運用開始年の1月の始業とともに運用が開始されている。3系統とも、昨年1年間の運用時間は6,240時間であった。
- (2) Xは、導入した年から数えて（導入した年を1年目として）13年経過し、14年目に入った回線であり、今年の12月いっぱいまで運用をやめることになっている。一昨年（12年目の1年間）と昨年（13年目の1年間）の故障頻度は、安定的に推移していたそれまでの数年間より増える傾向にあった。Xの昨年のMTTR（平均修復時間）は6時間であり、故障時間の合計はYの6倍であった。
- (3) Yは、導入後7年目の回線であり、3年前（4年目）から故障頻度は安定している。Yの昨年（6年目の1年間）のMTTRは4時間であり、故障回数は3回であった。
- (4) Zは、導入後2年目の最も新しい回線であり、Zの昨年（1年目の1年間）のMTTRは6時間であり、故障頻度はYより高い値であった。

〔バスタブ曲線の説明〕

ハードウェアの故障頻度は、横軸に導入からの経過時間、縦軸に故障頻度をとると、図のようなバスタブを縦に切った断面のような形状の曲線に従うことが知られており、この曲線をバスタブ曲線と呼んでいる。

バスタブ曲線では、運用開始直後から時間の経過とともに故障頻度が減少する期間を初期故障期間、老朽化によって故障頻度が増加する期間を摩耗故障期間、この二つの期間の間で、偶発故障が主になり故障頻度が安定する期間を偶発故障期間と呼ぶ。

X、Y、Zの3系統の回線の故障頻度は、同じバスタブ曲線に従うとする。

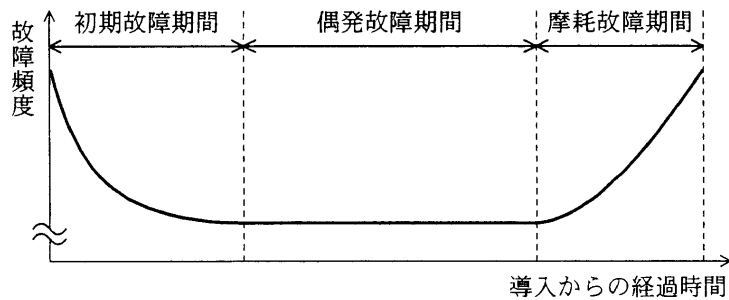


図 バスタブ曲線

設問 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。解答は、重複して選んでもよい。

バスタブ曲線の特徴から、YとZの故障頻度が両方ともに安定する時期は、今年から数えて a 年目から、 b 年間であるといえる。

表中の昨年のXのMTBF（平均故障間隔）は、 c 時間であり、Yの故障時間の合計は、 d 時間である。

MTBF、MTTRは、それぞれ次式で求められる。

$$\text{MTBF} = \text{稼働時間の合計} / \text{故障回数}$$

$$\text{MTTR} = \text{故障時間の合計} / \text{故障回数}$$

表 昨年の回線X, Y, Zの運用状況

回線	運用時間	MTTR (平均修復時間)	故障回数	故障時間の合計	MTBF (平均故障間隔)
X	6,240	6			<input type="text"/> c
Y	6,240	4	3	<input type="text"/> d	
Z	6,240	6			

注 網掛けの部分は、表示していない。

a, bに関する解答群

ア 1 イ 2 ウ 3 エ 4 オ 5

cに関する解答群

ア 72 イ 514 ウ 516 エ 520 オ 2,074

dに関する解答群

ア 1.3 イ 12 ウ 1,558 エ 2,080 オ 6,228

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。

[プログラムの説明]

スタックを使って、実数値を10進数字列(文字列)に変換する副プログラム FloatFormat である。

- (1) FloatFormat は、実数 Float の値を10進数字列に変換し、その先頭の数字から順に1文字ずつ文字型配列 Out[] に格納する。
- (2) 小数点以下は、引数 Num ($\text{Num} \geq 1$) で指定されたけた数までを格納する。
- (3) FloatFormat の引数の仕様を表に示す。

表 FloatFormat の引数の仕様

引数	データ型	入力/出力	意味
Float	実数型	入力	変換対象の実数
Num	整数型	入力	文字列に変換する際の小数点以下のけた数 ($\text{Num} \geq 1$) 小数第 Num + 1 位以下は切捨て
Out[]	文字型	出力	変換結果を格納する文字型の配列
Len	整数型	出力	配列 Out[] に格納された変換結果の文字数

- (4) 実数 Float の値を10進数字列に変換する手順は、次のとおりである。
 - ① Float の値が負の場合は、負符号を表す“-”を Out[] に格納し、Float の値を正数に変換する。
 - ② 整数部を、1の位から上位に向かって、1けたずつ10進数字に変換し、スタックに積む。
 - ③ スタックに積み終わったら、スタックに積んだ文字を順番に取り出して Out[] に格納することによって、整数部の10進数字を正しい順番に並べ替える。
 - ④ 整数部が0の場合は“0”を Out[] に格納する。
 - ⑤ 小数点を表す“.”を Out[] に格納する。
 - ⑥ 小数部を、小数第1位から第 Num 位まで、1けたずつ10進数字に変換し、Out[] に格納する。

- (5) `Push()` はスタックに1文字を積む関数, `Pop()` はスタックから1文字を取り出す関数である。`Int()` は小数点以下を切り捨てる関数である。
- (6) 配列の添字は0から始まり, 文字型配列 `Out[]` の要素数は十分に大きいものとする。また, プログラム中の各演算であふれは発生しないものとする。
- (7) `FloatFormat` の変換例を図に示す。

Float	Num	Out[]							
		0	1	2	3	4	5	6	...
12.345	3	1	2	.	3	4	5		
34.567	2	3	4	.	5	6			
-12.345	1	-	1	2	.	3			
0.012	2	0	.	0	1				
0.012	1	0	.	0					
-0.012	1	-	0	.	0				

図 FloatFormat の変換例

[プログラム]

```
○FloatFormat(実数型: Float, 整数型: Num, 文字型: Out[], 整数型: Len)
○実数型: F, Fdec
○整数型: Fint, Idx, L, N
○文字型: Chr[] = {"0","1","2","3","4","5","6","7","8","9"}, T
  ·L ← 0
/* 符号の処理 */
▲ Float ≥ 0.0
  ·F ← Float
  ───┬───
  │   │
  │   ·F ← -Float
  │   ·Out[0] ← "-"
  │   ·L ← 1
  ▼
/* 整数部の処理 */
·Push("#")          /* スタックに番兵として"#"を積む。 */
·Fint ← Int(F)
■ Fint > 0
  ·Idx ← Fint - (Fint ÷ 10) × 10 ← α
  ·Push(Chr[Idx])
  ·Fint ← a
■
·T ← Pop()          /* スタックから取り出す。 */
■ T ≠ "#"
  ·Out[L] ← T
  ·L ← L + 1
  ·T ← Pop()
■
/* 小数点の処理 */
▲ b
  ·Out[L] ← "0"
  ·L ← L + 1
  ▼
·Out[L] ← "."
·L ← L + 1
/* 小数部の処理 */
·Fdec ← F - Int(F)
■ N: 1, N ≤ Num, 1
  ·Fdec ← c ← β
  ·Idx ← Int(Fdec)
  ·Out[L] ← Chr[Idx]
  ·L ← L + 1
  ·Fdec ← d
■
·Len ← L
```

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 `Fint - 10`
- イ `Fint - Fint ÷ 10`
- ウ `Fint - (Fint ÷ 10) × 10`
- エ `Fint ÷ 10`
- オ `(Fint ÷ 10) × 10`

bに関する解答群

- ア `L = 0`
- イ `L = 0 or Out[0] = "-"`
- ウ `L = 0 or (L = 1 and Out[0] = "-")`
- エ `L = 1`
- オ `L = 1 and Out[0] = "-"`

c, dに関する解答群

- ア `Fdec - Int(Fdec)`
- イ `Fdec - Int(Fdec × 10)`
- ウ `Fdec - Int(Fdec ÷ 10)`
- エ `Fdec × 10`
- オ `Fdec ÷ 10`

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。

`Float = -0.012, Num = 2` として `FloatFormat` を呼び出した場合、
プログラム中の α の部分は `e` 回実行され、 β の部分は `f` 回
実行される。

解答群

- ア 0 イ 1 ウ 2 エ 3 オ 4

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。

Z社はWebを使った通信販売システムによって、会員に商品を販売している。会員は、Z社の提供する通信販売システムを利用して商品を購入する。今回、Z社では、会員ごとの購入実績を用いて、購入の可能性が高いと判定された商品を表示する関連商品表示プログラムを開発した。

[通信販売システムの概要]

- (1) 通信販売システムはZ社の会員だけが利用でき、会員は一意的な会員IDをもつ。
- (2) 通信販売対象の商品は商品表に登録されている。商品表のレコード様式を図1に示す。

商品表

商品コード	商品名	商品カテゴリ	登録日時	商品価格
-------	-----	--------	------	------

注 下線はキー項目を表す。

図1 商品表のレコード様式

- (3) 会員は次の手順で、商品の購入操作を行う。
 - ① 通信販売システムに会員IDとパスワードを入力し、ログインする。
 - ② 購入を希望する商品を検索する。
 - ③ 購入する商品を指定し、購入数量を入力する。商品価格と購入数量の乗算が行われ、購入金額に加えられる。購入する商品がほかにもある場合は、②に戻る。
 - ④ 購入金額を確認し、支払方法を入力する。このときの日時が購入日時として記録される。
 - ⑤ ログアウトする。
- (4) 会員が支払方法を入力したとき、商品の購入が成立し、その際に一意な購入番号を割り当てる。購入番号をキーとして、購入日時、会員ID及び購入金額を、購入表に登録する。また、会員が購入した商品は、商品ごとに購入番号と商品コードをキー項目として購入明細表に登録する。

なお、1回のログインで同一商品を複数回に分けて指定したときには、その商品の購入数量を合計し、1件のレコードとして購入明細表に登録する。購入表と購入明細表のレコード様式を図2に示す。

購入表

購入番号	購入日時	会員 ID	購入金額
------	------	-------	------

購入明細表

購入番号	商品コード	購入数量
------	-------	------

注 下線はキー項目を表す。

図 2 購入表と購入明細表のレコード様式

[関連商品表示プログラムの概要]

(1) 新規登録商品の表示機能

会員がログインしたとき、その会員 ID で最近購入した商品と同一商品カテゴリの商品の情報を、商品表から最大 10 件表示する機能である。ただし、表示する商品の情報は、その会員の最も新しい購入日時以降に商品表に登録されたものだけとする。

① 最新購入実績の抽出

ログイン時に入力された会員 ID を用いて、購入表と購入明細表からその会員の最も新しい購入の実績をすべて取り出す。

② 商品カテゴリの抽出

① で取り出した購入の実績のレコードと商品表のレコードを、商品コードをキーとして突き合わせ、一致したときに商品表から商品カテゴリを取り出す。

③ 登録商品の抽出

② で取り出した商品カテゴリのすべてと、商品表の商品カテゴリを突き合わせ、等しい商品カテゴリをもつ商品表のレコードをすべて取り出す。

④ 新規登録商品の表示

③ の結果を基に、その会員の最も新しい購入日時以降に商品表に登録された商品のうち、登録日時が新しいものから最大 10 件の情報を表示する。

(2) 検索した商品に関連する商品の表示機能

会員がある商品を検索したとき、その商品を既に購入した他会員が同時に購入した商品の情報を、最大 10 件表示する機能である。ただし、表示する商品の情報はその会員がまだ購入していない商品だけとする。

① 購入番号の抽出

検索した商品の商品コードを用いて、購入明細表からその商品コードをもつすべてのレコードの購入番号を取り出す。その購入番号とログイン時に入力された会員 ID を用いて、購入表を検索し、他会員の購入番号だけを取り出す。

② 同時購入商品の抽出

① で取り出した他会員の購入番号ごとに購入明細表を検索し、その購入番号をもつレコードをすべて取り出す。

③ 検索した商品の削除

② で取り出したレコードの商品コードと、会員が検索した商品の商品コードを突き合わせ、検索した商品の商品コードをもつレコードをすべて取り除く。

④ 関連商品の抽出

③ で残ったレコードの商品コードのすべてと商品表のレコードを、商品コードをキーとして突き合わせ、一致したレコードを商品表から取り出す。

⑤ 購入済商品の削除

④ の結果を基に、その会員が既に購入した商品の商品コードをもつレコードをすべて取り除く。

⑥ 関連商品の表示

⑤ の結果を基に、登録日時が新しいものから最大 10 件の商品の情報を取り出し表示する。

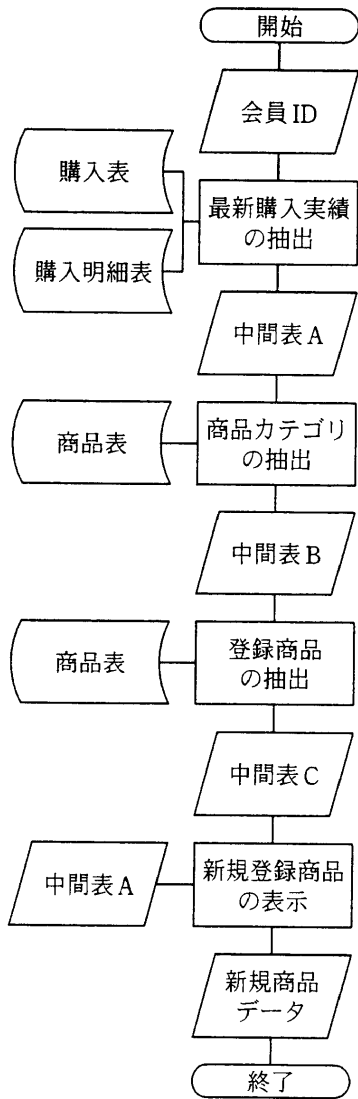


図3 新規登録商品の表示機能

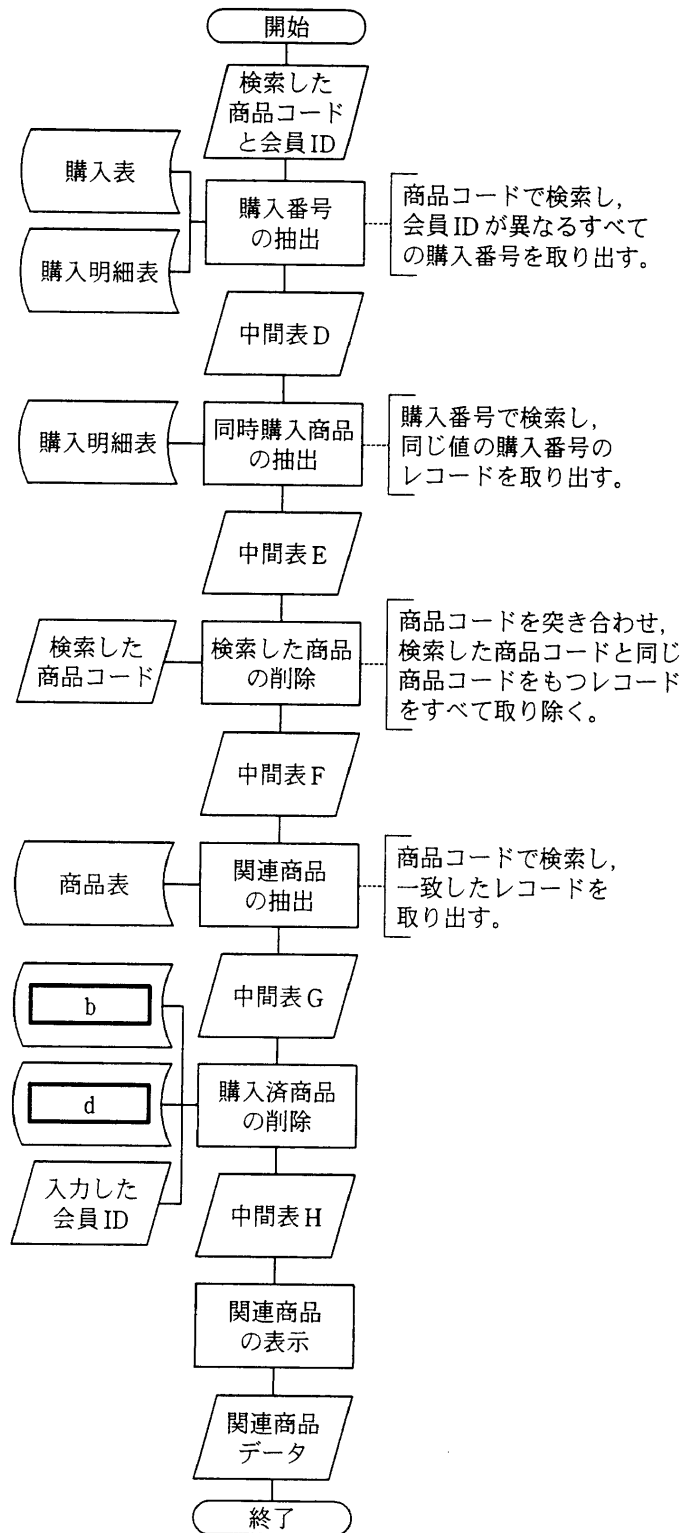


図4 検索した商品に関連する商品の表示機能

設問1 図3中の中間表A及び中間表Bで保持する項目のうち，“商品カテゴリの抽出”，“登録商品の抽出”で必要とする項目の組合せとして正しい答えを，解答群の中から選べ。

解答群

	中間表A	中間表B
ア	会員ID	商品カテゴリ
イ	会員ID	商品名
ウ	商品コード	商品価格
エ	商品コード	商品カテゴリ
オ	商品コード	商品名

設問2 図3中の“最新購入実績の抽出”に関する次の説明中の に入れる正しい答えを，解答群の中から選べ。

ログイン時に入力された会員IDを用いて，購入表からその会員の を1件取り出す。取り出したレコードの購入番号をもつレコードを購入明細表からすべて取り出す。取り出した結果を中間表Aのレコード様式に編集し，中間表Aに出力する。購入表に該当するレコードが存在しない場合は，何も出力しない。

解答群

- ア 最も新しい購入日時をもつレコード
- イ 最も多い購入数量をもつレコード
- ウ 最も大きい購入金額をもつレコード
- エ 最も小さい購入金額をもつレコード
- オ 最も古い購入日時をもつレコード

設問3 図4中の“購入済商品の削除”に関する次の説明中の□に入る正しい答えを、解答群の中から選べ。

中間表Gから読み込んだレコードの商品コードを用いて、□ b □を検索し、等しい商品コードをもつ□ c □をすべて取り出す。□ c □ごとに□ d □を検索し、ログインのときに入力した会員IDをもつレコードが一つでもあれば、商品は既に購入済みと判定し、取り除く。

b, dに関する解答群

ア 購入表

イ 購入明細表

ウ 商品表

cに関する解答群

ア 会員ID

イ 購入番号

ウ 商品コード

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]

引数の配列inに格納された8文字×6文字の文字パターンを、引数invtypeの値(1～4)に従って、回転又は反転して標準出力に出力する関数invertである。

(1) 引数invtypeの値と文字パターンの回転又は反転種別の関係は、表のとおりである。

表 引数invtypeの値と文字パターンの回転・反転種別

invtypeの値	回転・反転種別
1	90度右回転
2	90度左回転
3	上下反転
4	左右反転

(2) 関数mainの実行結果をまとめた文字パターンの印字例を図に示す。

元のパターン	90度右回転	90度左回転	上下反転	左右反転
#####	#####	#	#	#####
#	# #	#	#	#
#	# #	#	#	#
#	#	# #	###	#
###	#	# #	#	###
#	#	#####	#	#
#			#	#
#			#####	#

図 文字パターンの印字例

(3) 回転及び反転操作の手順を、8行6列の配列dat[8][6]を例として説明する。

90度回転では、回転後は行と列の数が入れ替わることになる。

90度右回転の場合には、次に示すように先頭の行データ群は、右端の列データ

群に移る。次の行データ群が右から二つ目の列に移り、同様の手順で移動が進み、最後に最下段の行データ群が左端の列データ群に移る。

```
先頭行： dat[0][0], dat[0][1], ..., dat[0][4], dat[0][5]
           ↓           ↓           ↓           ↓
右端列： dat[0][7], dat[1][7], ..., dat[4][7], dat[5][7]
```

90度左回転の場合には、先頭の行データ群は、左端の列データ群 $\text{dat}[5][0]$, $\text{dat}[4][0]$, ..., $\text{dat}[1][0]$, $\text{dat}[0][0]$ に移る。次の行データ群が左から二つ目の列に移り、同様の手順で移動が進み、最後に最下段の行データ群が右端の列データ群に移る。

上下反転では、先頭の行データ群は、最下段の行データ群 $\text{dat}[7][0]$, $\text{dat}[7][1]$, ..., $\text{dat}[7][4]$, $\text{dat}[7][5]$ に移る。次の行データ群が下から二つ目の行に移り、同様の手順で移動が進み、最後に最下段の行データ群が先頭の行データ群に移る。

左右反転では、行内で入れ替わるので、先頭の行データ群は、 $\text{dat}[0][5]$, $\text{dat}[0][4]$, ..., $\text{dat}[0][1]$, $\text{dat}[0][0]$ に移る。以降の行データ群も同様の手順で行内で順序が入れ替わる。

[プログラム]

```
#include <stdio.h>
#define RSZ 6
#define CSZ 8

void invert(int, char[][]);

void invert(int invtype, char in[CSZ][RSZ]){
    int row = RSZ, col = CSZ;
    int i, j;

    if(a){ /* 行数と列数の入替え */
        row = CSZ;
        col = RSZ;
    }
    for(i = 0; i < col; i++){
        for(j = 0; j < row; j++){
            switch(invtype){
                case 1: /* 90度右回転 */
                    printf("%c", in[CSZ - j - 1][i]);
                    break;
                case 2: /* 90度左回転 */
                    printf("%c", b);
                    break;
                case 3: /* 上下反転 */
                    printf("%c", in[CSZ - i - 1][j]);
                    break;
                case 4: /* 左右反転 */
                    printf("%c", c);
                    break;
                default:
                    break;
            }
        }
        printf("\n");
    }
}

void main(){
    char src[CSZ][RSZ] = {{'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'},
                          {'#', '#', '#', '#', '#', '#'}};

    invert(1, src);
    invert(2, src);
    invert(3, src);
    invert(4, src);
}
```


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 `invtype == 1`
- イ `invtype == 2`
- ウ `invtype == 3`
- エ `invtype == 4`
- オ `(invtype == 1) || (invtype == 2)`
- カ `(invtype == 3) || (invtype == 4)`

b, cに関する解答群

- ア `in[RSZ - i - 1][j]`
- イ `in[RSZ - j - 1][i]`
- ウ `in[i][CSZ - j - 1]`
- エ `in[i][RSZ - j - 1]`
- オ `in[j][CSZ - i - 1]`
- カ `in[j][RSZ - i - 1]`



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〕

ある会社では、1年に1回、従業員の定期健康診断を実施し、過去5年分の診断結果をマスタファイルに保管している。

このプログラムは、別のファイルに記録された今年度の診断結果を、マスタファイルに反映する。

(1) マスタファイルは、次のレコード様式の索引ファイルであり、主レコードキーは従業員番号である。

従業員番号 6けた	診断結果（1年前）				診断結果（2年前）				
	身長 4けた	体重 4けた	最高血圧 3けた	最低血圧 3けた	身長 4けた	体重 4けた	最高血圧 3けた	最低血圧 3けた	

診断結果を5回繰返し

- ① 従業員1人に対して1レコードとする。従業員番号が重複することはない。
 - ② レコードには、過去5年分の診断結果が左から受診年の新しい順で格納されている。各レコードには少なくとも過去1年分の診断結果が格納されており、就業年数が5年に満たない場合は残りの領域に空白が詰められている。
 - ③ 診断結果の各項目はいずれも数字で、身長はcm単位、体重はkg単位で、それぞれ小数点以下1けたまで記録され、血圧はmmHg単位の整数として記録されている。
- (2) 今年度の診断結果は、次のレコード様式の順ファイルであり、すべての従業員の情報が順不同で記録されている。同じ従業員番号のレコードが重複して存在することはない。

従業員番号 6けた	今年度の診断結果			
	身長 4けた	体重 4けた	最高血圧 3けた	最低血圧 3けた

[プログラム]

(行番号)

```
1 DATA DIVISION.
2 FILE SECTION.
3 FD MAST-FILE.
4 01 MAST-REC.
5     02 MAST-ENO          PIC X(6).
6     02 MAST-HEALTH      OCCURS 5.
7         03 MAST-HEIGHT  PIC 9(3)V9(1).
8         03 MAST-WEIGHT  PIC 9(3)V9(1).
9         03 MAST-HIGH-BP PIC 9(3).
10        03 MAST-LOW-BP  PIC 9(3).
11 FD YEAR-FILE.
12 01 YEAR-REC.
13     02 YEAR-ENO          PIC X(6).
14     02 YEAR-HEALTH.
15         03 YEAR-HEIGHT  PIC 9(3)V9(1).
16         03 YEAR-WEIGHT  PIC 9(3)V9(1).
17         03 YEAR-HIGH-BP PIC 9(3).
18         03 YEAR-LOW-BP  PIC 9(3).
19 WORKING-STORAGE SECTION.
20 77 READ-FLAG          PIC X(1) VALUE SPACE.
21     88 YEAR-EOF      VALUE "E".
22 77 CNT                PIC 9(1).
23 PROCEDURE DIVISION.
24 MAIN-PROC.
25     OPEN I-O MAST-FILE INPUT YEAR-FILE.
26     PERFORM UNTIL YEAR-EOF
27         READ YEAR-FILE AT END SET YEAR-EOF TO TRUE
28         NOT AT END PERFORM READ-MAST-PROC
29     END-READ
30     END-PERFORM.
31     CLOSE MAST-FILE YEAR-FILE.
32     STOP RUN.
33 READ-MAST-PROC.
34     [ a ].
35     READ MAST-FILE
36         INVALID KEY DISPLAY "ERROR EMPLOYEE-NO " YEAR-ENO
37         NOT INVALID KEY PERFORM UPDT-PROC
38     END-READ.
39 UPDT-PROC.
40     PERFORM VARYING CNT FROM 4 BY -1 UNTIL CNT = 0
41     [ b ].
42     END-PERFORM.
43     MOVE YEAR-HEALTH TO MAST-HEALTH(1).
44     REWRITE MAST-REC.
```

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。

解答群

- ア MOVE MAST-HEALTH(CNT + 1) TO MAST-HEALTH(CNT)
- イ MOVE MAST-HEALTH(CNT - 1) TO MAST-HEALTH(CNT)
- ウ MOVE MAST-HEALTH(CNT) TO MAST-HEALTH(CNT + 1)
- エ MOVE MAST-HEALTH(CNT) TO MAST-HEALTH(CNT - 1)
- オ MOVE YEAR-ENO TO MAST-ENO
- カ MOVE YEAR-HEALTH TO MAST-HEALTH
- キ MOVE YEAR-HEALTH TO MAST-HEALTH(CNT)

設問2 今年度の診断結果をマスタファイルに反映するときに、昨年の体重と比較して5 kg以上の増減がある従業員をチェックし、対象者の従業員番号を表示するように、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
<input type="text" value="c"/> に追加	<pre> IF YEAR-WEIGHT >= MAST-WEIGHT(1) + 5 OR <input type="text" value="d"/> THEN DISPLAY "HEALTH CHECK EMPLOYEE-NO " YEAR-ENO END-IF </pre>

cに関する解答群

- ア 行番号 26 と 27 の間
- イ 行番号 30 と 31 の間
- ウ 行番号 37 と 38 の間
- エ 行番号 39 と 40 の間

dに関する解答群

- ア MAST-WEIGHT(1) <= MAST-WEIGHT(2) - 5
- イ MAST-WEIGHT(1) >= MAST-WEIGHT(2) - 5
- ウ YEAR-WEIGHT <= MAST-WEIGHT(1) - 5
- エ YEAR-WEIGHT >= MAST-WEIGHT(1) - 5

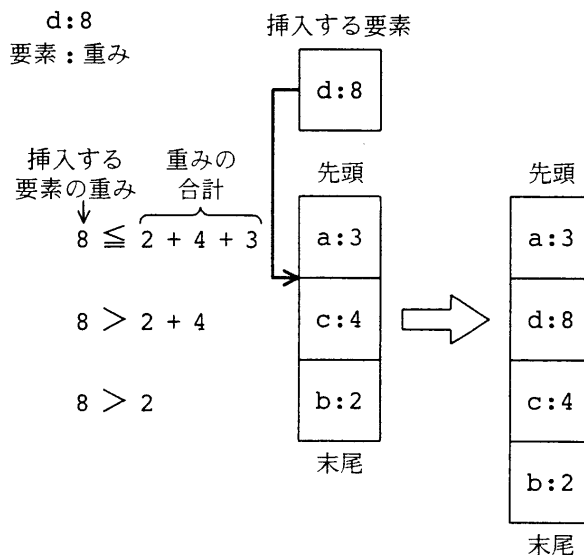
問8 次のJavaプログラムの説明及びプログラムを読んで、設問1, 2に答えよ。

(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 `WeightedQueue` は、次の規則に従う待ち行列を実現するプログラムである。

- (1) 待ち行列に入る要素には、重みが付けられる。重みは正の整数とする。ただし、この待ち行列に `null` を挿入することはできない。
- (2) 要素の挿入位置は、末尾の要素からの重みの合計が、挿入する要素の重み未満で、最も先頭寄りとする。ただし、重みの合計が `Long.MAX_VALUE` を超えたときの動作は保証しない。要素を挿入する例を図1に示す。



- (3) 要素を取り出すときは、その時点で先頭にある要素が取り出される。

[待ち行列の使用例]

この待ち行列に、優先度の高いタスク（重みの値が大きい要素）を挿入すると、このタスクは先に挿入されていた幾つかの優先度の低いタスクよりも、先に取り出されるようになる。この性質を利用して、この待ち行列を、優先度に応じたタスクのスケジューリングに使うことができる。

テスト用クラス `WeightedQueueTester` の実行結果を図2に示す。

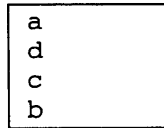


図2 実行結果

[プログラム1]

```
import java.util.LinkedList;

public class WeightedQueue<E> {
    private final LinkedList<QueueElement> queue =
        new LinkedList<QueueElement>();

    // 要素を待ち行列に挿入する。(element:要素, weight:重み)
    public void offer(E element, long weight) {

        if (element == null) {
            throw new NullPointerException();
        }

        int pos;
        long sum = 0;
        // 末尾の要素からの重みの合計がweight以上になるか、先頭に達するまで
        // 前方に向かって走査する。待ち行列の先頭位置は0
        for (pos = queue.size(); pos > 0; pos--) {
            sum += queue.get(a).weight;
            if (sum b weight) {
                break;
            }
        }
        queue.add(pos, c);
    }

    public E poll() {
        QueueElement e = queue.poll();
        if (e == null) {
            return null;
        } else {
            return e.element;
        }
    }
}
```

```
// 要素と重みの組
private class QueueElement {
    private final E element;
    private final long weight;

    private QueueElement(E element, long weight) {
        this.element = element;
        this.weight = weight;
    }
}
}
```

[プログラム2]

```
public class WeightedQueueTester {
    public static void main(String[] args) {
        WeightedQueue<String> wq = new WeightedQueue<String>();
        String[] s = {"a", "b", "c", "d"};
        long[] w = {3, 2, 4, 8};
        for (int i = 0; i < s.length; i++) {
            wq.offer(s[i], w[i]);
        }
        String data;
        while ((data = wq.poll()) != null) {
            System.out.println(data);
        }
    }
}
```

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。

JAVA

aに関する解答群

ア pos	イ pos - 1	ウ pos + 1
エ sum	オ sum - 1	カ sum + 1

bに関する解答群

ア !=	イ <=	ウ ==	エ >=
------	------	------	------

cに関する解答群

ア new QueueElement(element, sum)
イ new QueueElement(element, weight)
ウ QueueElement(element, sum)
エ QueueElement(element, weight)

設問2 クラス `WeightedQueue` のメソッド `poll` に関する記述として正しいものを、
解答群の中から選べ。

解答群

- ア 待ち行列の先頭要素の取得及び削除をする。ただし、待ち行列に要素がないときは `null` を返す。
- イ 待ち行列の先頭要素の取得及び削除をする。ただし、待ち行列に要素がないときは例外を投げる。
- ウ 待ち行列の先頭要素を取得する。要素の削除はしない。ただし、待ち行列に要素がないときは `null` を返す。
- エ 待ち行列の先頭要素を取得する。要素の削除はしない。ただし、待ち行列に要素がないときは例外を投げる。

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]

連続する N 語のブロックに格納されている各語にパリティビットを設定するとともに、ブロックの直後に1語の水平パリティを設定する副プログラム SETPAR である。

(1) 各語のビット番号 15 (図の破線で囲まれた左端ビット) をパリティビットとして使用する。パリティビットの値は偶数パリティとし、各語のビット番号 0～14 中の1であるビットの個数が奇数のときは1, 偶数のときは0とする。

水平パリティの各ビットの値も偶数パリティとし、各語の対応するビット番号中の1であるビットの個数が奇数のときは1, 偶数のときは0とする。つまり、図の網掛け部分において、1であるビットの個数が偶数となるように、水平パリティ中のビットを設定する。

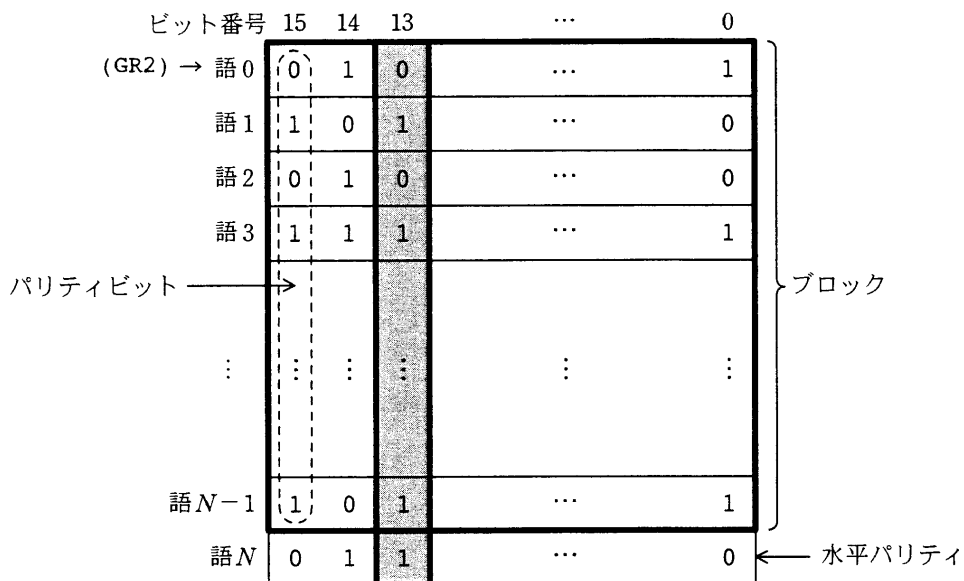


図 パリティビットと水平パリティ

- (2) 主プログラムは、ブロックの語数 N をGR1に、ブロックの開始アドレスをGR2に設定し、ブロックの各語のパリティビットに0を格納して副プログラム SETPAR を呼ぶ。
- (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。

[プログラム]

(行番号)

```
1  SETPAR  START
2          RPUSH
3          LD    GR6,GR2
4          ADDL  GR6,GR1
5          LAD   GR4,0          ; 水平パリティ初期化
6  LP1     LD    GR1,0,GR2     ; ブロックから1語取出し
7          LAD   GR7,0
8  LP2     SLL   GR1,1
9          JZE   CONT
10         
11         XOR   GR7,=#8000    ; パリティビット調整
12         JUMP  LP2
13  CONT   OR    GR7,0,GR2
14         ST    GR7,0,GR2    ; パリティビット設定
15          ; 水平パリティ調整
16         LAD   GR2,1,GR2
17         CPL   GR2,GR6      ; ブロックの終端?
18         JNZ   LP1
19         ST    GR4,0,GR6    ; 水平パリティ設定
20         RPOP
21         RET
22         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	JMI	CONT	イ	JMI	LP2	ウ	JOV	CONT
エ	JOV	LP2	オ	JPL	CONT	カ	JPL	LP2

bに関する解答群

ア	OR	GR4,0,GR6	イ	OR	GR4,GR7
ウ	OR	GR7,0,GR6	エ	OR	GR7,GR4
オ	XOR	GR4,0,GR6	カ	XOR	GR4,GR7
キ	XOR	GR7,0,GR6	ク	XOR	GR7,GR4

アセンブラ

設問2 パリティビットの値として、各語のビット番号0～14中の1であるビットの個数が奇数のときは0を、偶数のときは1を設定するようにプログラムを変更する。これは、プログラム中の行番号7でGR7に設定する値を変更すれば可能である。GR7に設定する値として正しい答えを、解答群の中から選べ。

解答群

ア #0001

イ #7FFF

ウ #8000

エ #8001

オ #FFFE

カ #FFFF

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]

関数 `calc_carriage` は、直方体の配送物の配送種別及び配送料金を求めるプログラムである。

(1) 関数 `calc_carriage` の引数は、次のとおりである。

`weight` : 配送物の重量 (単位はグラム)

`size` : 配送物の大きさ (縦, 横, 高さの合計, 単位はcm)

`dist_mode` : 配送先までの距離区分 (0 : 近距離, 1 : 中距離, 2 : 遠距離)

`class` : 配送種別

`price` : 配送料金 (単位は円)

(2) 配送種別は、配送物の重量及び大きさから、1, 2, 3, -1, -2 のいずれか一つの値に定まる。配送種別が -1 又は -2 の配送物は、受け付けない。

(3) 配送料金は、配送種別ごとに用意された計算式によって求められる。

[プログラム]

```
#define NUM1 3
#define NUM2 4
#define NUM3 3

/* 各重量範囲での上限重量 (規定サイズ内) */
const int Weight_tbl1[NUM1] = {25, 60, 100},
        Price_tbl1[NUM1]   = {50, 80, 150};

/* 各重量範囲での上限重量 (規定サイズ外) */
const int Weight_tbl2[NUM2] = {50, 100, 500, 1000};

const int Size_tbl[NUM3]    = {60, 100, 150},
        Mag_tbl[NUM3]      = {1, 3, 4};

const int Min_size = 20, Max_size = 40, Max_weight = 10000;
const int Unit_price = 2, Base_price = 600, Ext_price = 100;

void calc_carriage(int, int, int, int *, int *);

void calc_carriage(int weight, int size, int dist_mode,
        int *class, int *price){
    int i;

    if(weight <= Weight_tbl2[NUM2 - 1]){
        if((size >= Min_size) && (size <= Max_size) &&
            (weight <= Weight_tbl1[NUM1 - 1])){
            *class = 1;
            for(i = 0; weight > Weight_tbl1[i]; i++);
            *price = Price_tbl1[i];
        }else{
            *class = 2;
            for(i = 0; weight > Weight_tbl2[i]; i++);
            *price = Weight_tbl2[i] * Unit_price;
        }
    }else if(weight <= Max_weight){
        if(size <= Size_tbl[NUM3 - 1]){
            *class = 3;
            for(i = 0; size > Size_tbl[i]; i++);
            *price = (Base_price + dist_mode * Ext_price) * Mag_tbl[i];
        }else{
            *class = -1;
            *price = 0;
        }
    }else{
        *class = -2;
        *price = 0;
    }
}
}
```

設問1 重量が1,000グラム ($Weight_tbl2[NUM2 - 1]$) を超えて10,000グラム (Max_weight) 以下の配送物の配送料金は表のとおりになる。表中の に入れる正しい答えを、解答群の中から選べ。

表 重量が1,000グラムを超えて10,000グラム以下の配送物の配送料金
単位 円

距離区分 ($dist_mode$)	近距離	中距離	遠距離
大きさ ($size$)			
60cm ($Size_tbl[0]$) 以下	600		800
60cm を超えて 100cm ($Size_tbl[1]$) 以下		<input type="text" value="a"/>	
100cm を超えて 150cm ($Size_tbl[2]$) 以下	2,400		3,200

注 網掛けの部分は、表示していない。

解答群

ア 1,600 イ 1,750 ウ 1,900 エ 2,100 オ 2,300

設問2 配送種別及び配送料金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。

- (1) 重量600グラム、大きさ180cmの配送物を送る場合、配送種別は となる。
- (2) 重量3,000グラム、大きさ180cmの配送物を送る場合、配送種別は となる。
- (3) 重量12,000グラム、大きさ180cmの配送物を送る場合、配送種別は となる。
- (4) 重量80グラム、大きさ30cmの配送物を「中距離」の場所へ送る場合、配送料金は 円となる。
- (5) 重量400グラム、大きさ30cmの配送物を「中距離」の場所へ送る場合、配送料金は 円となる。

b～dに関する解答群

ア -2 イ -1 ウ 1 エ 2 オ 3

e, fに関する解答群

ア 80 イ 150 ウ 250
エ 500 オ 1,000 カ 2,000



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]

B社は、取引先あてに請求書と振込用紙を発行し、取引先はその振込用紙を使ってC銀行に請求金額を振り込む。このプログラムは、毎月C銀行から受け取る振込データと、社内で管理している請求データの突合せを行って、入金確認リストを印字するものである。突合せキーは、一意に振られている請求書コードである。

(1) 銀行から受け取る振込データ BANK-Fは順ファイルで、レコード様式は、次のとおりである。レコードは、振込日で昇順に整列されている。

振込日 8けた	請求書コード 6けた	振込金額 9けた	取引先名 20けた
------------	---------------	-------------	--------------

(2) 社内で管理している請求データ ACCOUNT-Fは順ファイルで、レコード様式は、次のとおりである。レコードは、請求書コードで昇順に整列されている。

請求日 8けた	請求書コード 6けた	請求金額 9けた	取引先名 20けた
------------	---------------	-------------	--------------

(3) 入金確認リストの印字様式は、次のとおりである。

請求書コード	取引先名	入金予定金額	振込金額	差額	入金確認結果
XXXXXX	XXX ... XX	¥, ¥¥¥, ¥¥¥, ¥¥9	¥, ¥¥¥, ¥¥¥, ¥¥9	-, ---, ---, --9	XXX ... XX
XXXXXX	XXX ... XX	¥, ¥¥¥, ¥¥¥, ¥¥9	¥, ¥¥¥, ¥¥¥, ¥¥9	-, ---, ---, --9	XXX ... XX
⋮	⋮	⋮	⋮	⋮	⋮

- ① 入金確認リストの見出しは印刷済みとする。
- ② 入金予定金額には請求金額を印字する。
- ③ 入金確認結果は、次の4種類のいずれかを印字する。

印字メッセージ	意味
OK	入金予定金額と振込金額が一致
UNMATCH	金額不一致
NOT PAID	振込みなし
NO BILL	請求なし

[プログラム]

(行番号)

```
1 DATA DIVISION.
2 FILE SECTION.
3 FD BANK-F.
4 01 BANK-R PIC X(43).
5 SD SORT-F.
6 01 SORT-R.
7 05 BANK-DATE PIC 9(8).
8 05 BANK-BILL-CD PIC X(6).
9 05 BANK-AMOUNT PIC 9(9).
10 05 BANK-CUSTOMER-NM PIC X(20).
11 FD ACCOUNT-F.
12 01 ACCOUNT-R.
13 05 ACCOUNT-DATE PIC 9(8).
14 05 ACCOUNT-BILL-CD PIC X(6).
15 05 ACCOUNT-AMOUNT PIC 9(9).
16 05 ACCOUNT-CUSTOMER-NM PIC X(20).
17 FD PRINT-F.
18 01 PRINT-R PIC X(128).
19 WORKING-STORAGE SECTION.
20 01 W-BANK-BILL-CD PIC X(6).
21 01 W-ACCOUNT-BILL-CD PIC X(6).
22 01 P-DETAIL.
23 05 P-BILL-CD PIC X(6).
24 05 FILLER PIC X(6).
25 05 P-CUSTOMER-NM PIC X(20).
26 05 FILLER PIC X.
27 05 P-ACCOUNT-AMOUNT PIC ¥,¥¥¥,¥¥¥,¥¥9.
28 05 FILLER PIC X.
29 05 P-BANK-AMOUNT PIC ¥,¥¥¥,¥¥¥,¥¥9.
30 05 FILLER PIC X.
31 05 P-DIFF PIC -,---,---,--9.
32 05 FILLER PIC X.
33 05 P-MESSAGE PIC X(30).
34 PROCEDURE DIVISION.
35 MAIN-CTL.
36 SORT SORT-F ASCENDING KEY a
37 USING BANK-F
38 OUTPUT PROCEDURE MATCH-PROC.
39 STOP RUN.
40 MATCH-PROC.
41 OPEN INPUT ACCOUNT-F.
42 OPEN OUTPUT PRINT-F.
43 PERFORM RETURN-BANK.
44 PERFORM READ-ACCOUNT.
```

COBOL

```

45     PERFORM UNTIL W-BANK-BILL-CD    = HIGH-VALUE AND
46           W-ACCOUNT-BILL-CD = HIGH-VALUE
47     MOVE SPACE TO P-DETAIL
48     EVALUATE TRUE
49       WHEN W-BANK-BILL-CD > W-ACCOUNT-BILL-CD
50         MOVE ACCOUNT-CUSTOMER-NM TO P-CUSTOMER-NM
51         MOVE ACCOUNT-BILL-CD    TO P-BILL-CD
52         MOVE ACCOUNT-AMOUNT     TO P-ACCOUNT-AMOUNT
53         b
54         WRITE PRINT-R FROM P-DETAIL
55         PERFORM READ-ACCOUNT
56       WHEN W-BANK-BILL-CD = W-ACCOUNT-BILL-CD
57         MOVE BANK-CUSTOMER-NM TO P-CUSTOMER-NM
58         MOVE ACCOUNT-BILL-CD  TO P-BILL-CD
59         MOVE ACCOUNT-AMOUNT  TO P-ACCOUNT-AMOUNT
60         MOVE BANK-AMOUNT     TO P-BANK-AMOUNT
61         IF BANK-AMOUNT - ACCOUNT-AMOUNT = 0 THEN
62           MOVE "OK" TO P-MESSAGE
63         ELSE
64           c
65         END-IF
66         COMPUTE P-DIFF = BANK-AMOUNT - ACCOUNT-AMOUNT
67         WRITE PRINT-R FROM P-DETAIL
68         PERFORM READ-ACCOUNT
69         PERFORM RETURN-BANK
70       WHEN W-BANK-BILL-CD < W-ACCOUNT-BILL-CD
71         MOVE BANK-CUSTOMER-NM TO P-CUSTOMER-NM
72         MOVE BANK-BILL-CD     TO P-BILL-CD
73         MOVE BANK-AMOUNT     TO P-BANK-AMOUNT
74         d
75         WRITE PRINT-R FROM P-DETAIL
76         e
77     END-EVALUATE
78     END-PERFORM.
79     CLOSE ACCOUNT-F.
80     CLOSE PRINT-F.
81 *
82     RETURN-BANK.
83     RETURN SORT-F
84     AT END     MOVE HIGH-VALUE     TO W-BANK-BILL-CD
85     NOT AT END MOVE BANK-BILL-CD TO W-BANK-BILL-CD
86     END-RETURN.
87 *
88     READ-ACCOUNT.
89     READ ACCOUNT-F
90     AT END     MOVE HIGH-VALUE     TO W-ACCOUNT-BILL-CD
91     NOT AT END MOVE ACCOUNT-BILL-CD TO W-ACCOUNT-BILL-CD
92     END-READ.

```

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 BANK-AMOUNT

イ BANK-BILL-CD

ウ BANK-CUSTOMER-NM

エ BANK-DATE

b～eに関する解答群

ア MOVE "NO BILL" TO P-MESSAGE

イ MOVE "NOT PAID" TO P-MESSAGE

ウ MOVE "UNMATCH" TO P-MESSAGE

エ PERFORM READ-ACCOUNT

オ PERFORM RETURN-BANK

設問2 B社への振込手数料は、取引先が負担していたが、一部の大口取引に対しては、B社が振込手数料を負担することにした。この場合、銀行から受け取る振込金額は、請求金額から振込手数料を差し引いた金額となる。振込手数料の金額とその負担先は、請求金額によって決定される。振込手数料の計算には、次の様式の手数料計算プログラムCHARGEを使用する。

CALL "CHARGE" USING 請求金額 振込手数料

手数料計算プログラムは、請求金額を基に、振込手数料を返す。ただし、B社が負担しない場合はゼロを返す。

この対応のため、プログラムを変更する。入金確認リストの入金予定金額は請求金額から振込手数料を差し引いた金額を印字する。次の表中の に入れる正しい答えを、解答群の中から選べ。

処置	プログラムの変更内容
行番号33と34の間に追加	01 W-CHARGE PIC 9(6).
<input type="text"/> f 及び <input type="text"/> g に追加	CALL "CHARGE" USING ACCOUNT-AMOUNT W-CHARGE COMPUTE ACCOUNT-AMOUNT = ACCOUNT-AMOUNT - W-CHARGE

解答群

ア 行番号47と48の間

イ 行番号49と50の間

ウ 行番号56と57の間

エ 行番号70と71の間

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]

複数の文字列の並びから条件に合う文字列を抽出するプログラムである。

例えば、文字列の並びから特定の文字列で始まるものを抽出することができる。表に文字列の並びから条件に合う文字列を抽出する例を示す。

表 文字列の並びから条件に合う文字列を抽出する例

文字列の並び	条件	抽出結果
"one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten"	"t"で始まる文字列	"two", "three", "ten"

(1) プログラムは次のクラス及びインタフェースから構成される。

① クラス `Extraction<E>`

条件に当てはまる型 `E` の要素を抽出する。

コンストラクタの引数には、抽出元の要素の並びと抽出する条件を指定する。要素の並びは、`java.util.LinkedList` などの `java.lang.Iterable` を実装したクラスのインスタンスとして与える。

メソッド `iterator` は、抽出結果を取得するための反復子を返す。

② インタフェース `Evaluator<E>`

型 `E` の要素の抽出条件を定義するインタフェースである。

メソッド `evaluate` は、引数で指定された要素が抽出条件を満たしていれば `true` を、満たしていなければ `false` を返す。

(2) (1)のクラス及びインタフェースをテストするために次のクラスを用意した。

① クラス `StringStartsEvaluator`

指定された文字列で始まる文字列を抽出する条件を実装する。

コンストラクタの引数には、抽出される文字列の先頭文字列を指定する。

メソッド `evaluate` は、引数で指定された文字列が、コンストラクタで指定された文字列で始まるならば `true` を、それ以外は `false` を返す。

② クラス ExtractionTester

クラス StringStartsEvaluator を利用して、文字列の並びから "t" で始まる文字列を抽出し、表示する。

ExtractionTester の実行結果を図に示す。

```
two
three
ten
```

図 実行結果

[プログラム1]

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class Extraction<E> implements Iterable<E> {
    private final Iterable<E> target;
    private final Evaluator<E> evaluator;

    public Extraction(Iterable<E> target, Evaluator<E> evaluator) {
        this.target = target;
        this.evaluator = evaluator;
    }

    public Iterator<E> iterator() {
        return new ExtractionIterator();
    }

    // 抽出結果を取得するための反復子
    private class ExtractionIterator implements Iterator<E> {
        private final Iterator<E> iterator;
        private boolean found = false;
        private E element;

        private ExtractionIterator() {
            iterator = target.iterator();
        }

        // メソッド next でまだ取得されていない、抽出すべき要素があれば、
        // true を返す。
        public boolean hasNext() {
            // iterator.hasNext() が true ならば、抽出すべき要素が
            // まだあるかもしれない。
            while (!found & iterator.hasNext()) {
                // 抽出元の要素の並びから要素の一つ得る。
                element = iterator.next();
            }
        }
    }
}
```

```

        // 要素が抽出条件に合うか否かを検査する。
        // 条件に合えば found は true になる。
        found = evaluator.evaluate(element);
    }
    return ;
}

// 抽出された要素を一つずつ返す。
// 返すべき要素がないときは、例外を投げる。
public E next() {
    if (!hasNext()) {
        throw new NoSuchElementException();
    }
    // メソッド hasNext の次回の呼出しに備える。
    found = ;
    return ;
}

// この操作は提供しない。
public void remove() {
    throw new UnsupportedOperationException();
}
}
}

```

[プログラム2]

```

public interface Evaluator<E> {
    public boolean evaluate(E e);
}

```

[プログラム3]

```

public class StringStartsEvaluator
    implements Evaluator<String> {
    private final String prefix;

    public StringStartsEvaluator(String prefix) {
        if (prefix == null) {
            throw new NullPointerException();
        }
        this.prefix = prefix;
    }

    public boolean evaluate(String s) {
        return s != null && s.startsWith(prefix);
    }
}

```

[プログラム4]

```
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class ExtractionTester {
    public static void main(String[] args) {
        List<String> target =
            Arrays.asList("one", "two", "three", "four", "five",
                "six", "seven", "eight", "nine", "ten");
        Extraction<String> e = new Extraction<String>
            (target, new StringStartsEvaluator("t"));
        for (String s : e) {
            System.out.println(s);
        }
    }
}
```

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 != イ && ウ == エ ||

bに関する解答群

ア !found イ found
ウ element != null エ element == null

cに関する解答群

ア false イ hasNext()
ウ iterator.hasNext() エ true

dに関する解答群

ア E イ element
ウ iterator.next() エ next()

設問2 次のプログラム5に示す条件クラスを新たに作成した。この条件クラスで抽出される文字列として正しい答えを、解答群の中から二つ選べ。

[プログラム5]

```
public class AnEvaluator implements Evaluator<String> {
    public boolean evaluate(String s) {
        if (s != null) {
            final int len = s.length();
            for (int i = 0; i < len - 1; i++) {
                for (int j = i + 1; j < len; j++) {
                    if (s.charAt(i) == s.charAt(j)) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
}
```

解答群

ア "one"	イ "two"	ウ "three"	エ "four"
オ "five"	カ "six"	キ "seven"	ク "eight"

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]

連続する16語を16×16ビットからなる2次元配列とみなし、この配列の中で、1であるビットが16個並んでいる行、列及び対角線の本数を数える副プログラムBTESTである。図の例では網掛けの部分が該当し、本数は4である。

ビット番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(GR1)→語0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1
語1	0	1	0	0	1	0	1	0	0	1	0	0	1	0	1	0
語2	0	0	0	0	1	0	1	0	0	1	1	0	1	1	0	0
語3	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0
語4	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0
語5	0	0	1	0	1	1	1	1	0	0	1	0	1	1	1	0
語6	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1
語7	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0
語8	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0
語9	0	0	1	0	0	0	1	0	0	0	1	0	1	1	0	0
語10	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1	0
語11	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
語12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
語13	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0
語14	0	1	0	0	0	0	1	0	1	0	1	0	1	0	1	1
語15	1	0	0	0	0	0	1	0	1	1	0	0	1	0	1	1

図 ビットの配列例

- (1) 語0のアドレスがGR1に設定されて、主プログラムから渡される。
- (2) 本数を数えた結果はGR0に設定して、主プログラムに返す。
- (3) 行番号21で呼び出される副プログラムSETGR0は、副プログラムBTESTから渡されたGR0、GR2、GR3、GR4の内容を基に、列及び対角線の本数を加えた結果をGR0に設定する。
- (4) 副プログラムBTEST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1〕

(行番号)

```

1  BTEST  START
2      RPUSH
3      LD   GR0,=0
4      LD   GR2,=#0001 ; 左下がり対角線の検査用ビット
5      LD   GR3,=#8000 ; 右下がり対角線の検査用ビット
6      LD   GR4,=#FFFF ; 列の検査用ビット
7      LD   GR5,=16 ; ループカウンタ
8  LOOP1 LD   GR6,0,GR1
9      AND  GR2,GR6 ; } GR6 中で検査用ビットに
10     AND  GR3,GR6 ; } 対応するビットが0であれば
11     AND  GR4,GR6 ; } 検査用ビットを0にする。
12     CPL  GR6,=#FFFF ; 行のビットはすべて1か?
13     
14     ADDA GR0,=1 ; 行のカウンタ
15  NWORD SUBA GR5,=1 ; 全語処理済み?
16     
17     SLL  GR2,1 ; } 対角線の検査用ビットを
18     SRL  GR3,1 ; } 1ビットずつシフト
19     LAD  GR1,1,GR1 ; ポインタを次の語に位置付ける。
20     JUMP LOOP1
21  CHECK CALL  SETGR0 ; 列と対角線についてGR0に結果を設定
22     RPOP
23     RET
24     END

```

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。

解答群

ア JMI LOOP1 イ JNZ CHECK ウ JNZ NWORD
エ JPL LOOP1 オ JZE CHECK カ JZE NWORD

アセンブラ

設問2 配列の内容が図のとおりであった場合、行番号21のCALL命令実行直前における、次の表に示すレジスタの内容として正しい答えを、解答群の中から選べ。

レジスタ	内容
GR0	<input type="text" value="c"/>
GR2	<input type="text" value="d"/>
GR3	<input type="text" value="e"/>
GR4	<input type="text" value="f"/>

cに関する解答群

ア #0000 イ #0001 ウ #0002 エ #0003

d, eに関する解答群

ア #0000 イ #0001 ウ #8000 エ #8001

fに関する解答群

ア #0208 イ #8208 ウ #82CB エ #FFFF

設問3 副プログラム SETGR0 を次に示す。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。

〔プログラム2〕

```

SETGR0 START
LOOP2  SLL     GR4,1          ; 列の検査用ビットのうち1であるビットを数える。
      JOV     COUNT
               g
      JUMP    XCHECK
COUNT ADDA    GR0,=1        ; 列のカウント
      JUMP    LOOP2
XCHECK          h          ; 対角線の検査用ビットをシフト
      ADDA    GR0,GR2
      ADDA    GR0,GR3
      RET
      END

```

gに関する解答群

ア JMI LOOP2 イ JNZ LOOP2
ウ JPL LOOP2 エ JZE LOOP2

hに関する解答群

ア SLL GR2,14 イ SLL GR2,15
ウ SRL GR2,14 エ SRL GR2,15

■ Java プログラムで使用する API の説明

<code>java.lang</code> public interface Iterable<E> このインタフェースを実装すると、オブジェクトを拡張 for 文の対象にできる。
メソッド
----- public Iterator<E> iterator() 型 E の要素セットの反復子を返す。 戻り値：型 E の要素セットの反復子

<code>java.util</code> 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。 <code>java.lang.Iterable</code> を継承する。
メソッド
----- public Iterator iterator() このリスト内の要素を、適切な順序で繰り返し処理する反復子を返す。 戻り値：リスト内の要素を適切な順序で繰り返し処理する反復子

<pre>java.util public class LinkedList<E> インタフェース List のリンクリストを用いた実装である。</pre>
<p>コンストラクタ</p> <pre>public LinkedList() 空の LinkedList を作る。</pre>
<p>メソッド</p> <pre>public void add(int index, E element) リスト内の指定された位置に、指定された要素を挿入する。 引数： index — 指定した要素を挿入する位置（先頭は 0） element — 挿入する要素</pre> <pre>public E get(int index) リスト内の指定された位置にある要素を返す。 引数： index — 返される要素の位置（先頭は 0） 戻り値： リスト内の指定された位置にある要素</pre> <pre>public E poll() リストの先頭の要素を取り出す。取り出された要素はリストから削除される。 戻り値： リストの先頭の要素 リストが空なら null</pre> <pre>public int size() リスト内の要素の個数を返す。 戻り値： リスト内の要素の個数</pre>

<pre>java.util public interface Iterator<E> 要素を一つずつ参照する反復子のインタフェースを提供する。</pre>
<p>メソッド</p> <pre>public boolean hasNext() 未参照の要素があれば true を返す。 戻り値： 未参照の要素があれば true それ以外は false</pre> <pre>public E next() コレクションの要素のうち、このメソッドで未参照の要素の一つ返す。 戻り値： 未参照の要素の一つ 例外： NoSuchElementException — すべての要素が参照済みである。</pre> <pre>public void remove() メソッド next で最後に参照した要素を、基になるコレクションから削除する。この 操作は提供しなくてもよい。 例外： UnsupportedOperationException — 実装が remove を提供しない。</pre>

<pre>java.lang public final class String クラス String は文字列を表す。</pre>
<p>メソッド</p> <hr/> <pre>public char charAt(int index) 指定された位置にある char 型の値を返す。 引数： index — char 型の値の位置（先頭は 0） 戻り値： 指定された位置にある char 型の値</pre> <hr/> <pre>public int length() この文字列の長さを返す。 戻り値： このオブジェクトによって表される文字列の長さ</pre> <hr/> <pre>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</pre>

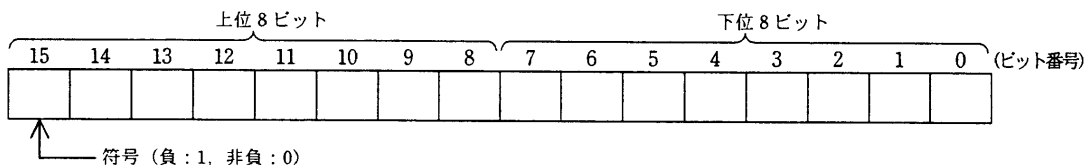
<pre>java.util public class Arrays クラス Arrays は、配列を操作するクラスメソッドからなる。</pre>
<p>メソッド</p> <hr/> <pre>public static <T> List<T> asList(T... a) 指定された配列を基にする固定サイズのリストを返す。 引数： a — 型 T の可変個の要素の並び 戻り値： 指定された要素のリスト（要素数は固定）</pre>

■アセンブラ言語の仕様

1. システム COMET II 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、それ以外るとき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、それ以外るとき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、それ以外るとき0になる。
ZF	演算結果が零 (全部のビットが0) のとき1、それ以外るとき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 r, adr [, x]	r1 ← (r2) r ← (実効アドレス)	○*1
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	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) + (r2)$ $r \leftarrow (r) + (\text{実効アドレス})$	○
論理加算 ADD Logical	ADDL	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) +_L (r2)$ $r \leftarrow (r) +_L (\text{実効アドレス})$	
算術減算 SUBtract Arithmetic	SUBA	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) - (r2)$ $r \leftarrow (r) - (\text{実効アドレス})$	
論理減算 SUBtract Logical	SUBL	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) -_L (r2)$ $r \leftarrow (r) -_L (\text{実効アドレス})$	
論理積 AND	AND	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) \text{ AND } (r2)$ $r \leftarrow (r) \text{ AND } (\text{実効アドレス})$	
論理和 OR	OR	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) \text{ OR } (r2)$ $r \leftarrow (r) \text{ OR } (\text{実効アドレス})$	
排他的論理和 eXclusive OR	XOR	$r1, r2$ $r, \text{adr} [,x]$	$r1 \leftarrow (r1) \text{ XOR } (r2)$ $r \leftarrow (r) \text{ XOR } (\text{実効アドレス})$	○*1

(3) 比較演算命令

算術比較 ComPare Arithmetic	CPA	$r1, r2$ $r, \text{adr} [,x]$	<p>(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。</p> <table border="1"> <thead> <tr> <th rowspan="2">比較結果</th> <th colspan="2">FR の値</th> </tr> <tr> <th>SF</th> <th>ZF</th> </tr> </thead> <tbody> <tr> <td>(r1) > (r2)</td> <td>0</td> <td>0</td> </tr> <tr> <td>(x) > (実効アドレス)</td> <td>0</td> <td>0</td> </tr> <tr> <td>(r1) = (r2)</td> <td>0</td> <td>1</td> </tr> <tr> <td>(r) = (実効アドレス)</td> <td>0</td> <td>1</td> </tr> <tr> <td>(r1) < (r2)</td> <td>1</td> <td>0</td> </tr> <tr> <td>(x) < (実効アドレス)</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	比較結果	FR の値		SF	ZF	(r1) > (r2)	0	0	(x) > (実効アドレス)	0	0	(r1) = (r2)	0	1	(r) = (実効アドレス)	0	1	(r1) < (r2)	1	0	(x) < (実効アドレス)	1	0	○*1
比較結果	FR の値																										
	SF	ZF																									
(r1) > (r2)	0	0																									
(x) > (実効アドレス)	0	0																									
(r1) = (r2)	0	1																									
(r) = (実効アドレス)	0	1																									
(r1) < (r2)	1	0																									
(x) < (実効アドレス)	1	0																									
論理比較 ComPare Logical	CPL	$r1, r2$ $r, \text{adr} [,x]$																									

(4) シフト演算命令

算術左シフト Shift Left Arithmetic	SLA	$r, \text{adr} [,x]$	<p>符号を除き (x) 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。</p> <p>符号を含み (x) 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。</p>	○*2
算術右シフト Shift Right Arithmetic	SRA	$r, \text{adr} [,x]$		
論理左シフト Shift Left Logical	SLL	$r, \text{adr} [,x]$		
論理右シフト Shift Right Logical	SRL	$r, \text{adr} [,x]$		

(5) 分岐命令

正分岐 Jump on Plus	JPL	$\text{adr} [,x]$	<p>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。</p> <table border="1"> <thead> <tr> <th rowspan="2">命令</th> <th colspan="3">分岐するときの FR の値</th> </tr> <tr> <th>OF</th> <th>SF</th> <th>ZF</th> </tr> </thead> <tbody> <tr> <td>JPL</td> <td></td> <td>0</td> <td>0</td> </tr> <tr> <td>JMI</td> <td></td> <td>1</td> <td></td> </tr> <tr> <td>JNZ</td> <td></td> <td></td> <td>0</td> </tr> <tr> <td>JZE</td> <td></td> <td></td> <td>1</td> </tr> <tr> <td>JOV</td> <td>1</td> <td></td> <td></td> </tr> </tbody> </table>	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			-
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																														
負分岐 Jump on Minus	JMI	$\text{adr} [,x]$																													
非零分岐 Jump on Non Zero	JNZ	$\text{adr} [,x]$																													
零分岐 Jump on Zero	JZE	$\text{adr} [,x]$																													
オーバフロー分岐 Jump on Overflow	JOV	$\text{adr} [,x]$																													
無条件分岐 unconditional JUMP	JUMP	$\text{adr} [,x]$	無条件に実効アドレスに分岐する。																												

(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

スーパーバイザコール 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 CASL II の仕様

2.1 言語の仕様

- (1) CASL II は、COMET II 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 (COMET II 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 $-32768 \sim 32767$ 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 ($0000 \leq h \leq FFFF$)。
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関係処理を行いアドレスを決定する（プログラムの関係）。
- (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]

6. 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
--------	---------------

7. 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。
8. 問題冊子の余白などは、適宜利用して構いません。
9. 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。
10. 電卓は、使用できません。
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。

試験問題に記載されている会社名又は製品名は、それぞれ各社の商標又は登録商標です。

なお、試験問題では、® 及び ™ を明記していません。