平成16年度　秋期

基本情報技術者
午後　問題

注意事項

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

試験時間	13:00 ～ 15:30（2時間30分）

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

退出可能時間	13:40 ～ 15:20

5. 問題は，次の表に従って解答してください。

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

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

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

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，この記述形式が適用されているものとする。

〔記述形式〕

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。
	■ **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　ディスク上のファイル管理システムに関する次の記述を読んで，設問1，2に答えよ。

図1のような階層的ディレクトリ構造を扱うファイル管理システムがある。

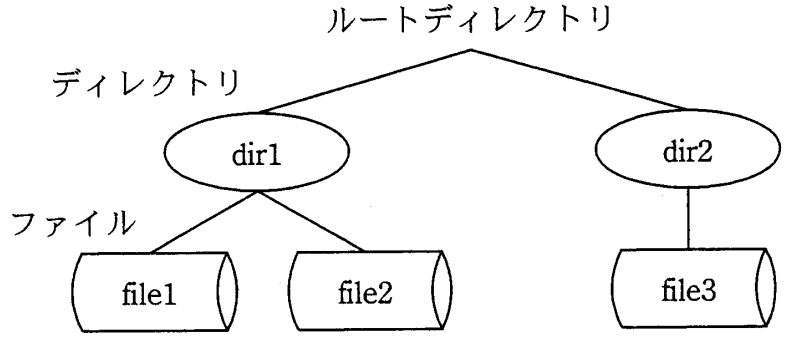


図1　ディレクトリ構造の例

このシステムでは，ファイルアロケーションテーブル（FAT）とディレクトリ管理テーブル（DCT）の二つのテーブルによって，ファイル及びディレクトリの情報を管理している。

(1) ディスクは，セクタと呼ばれる一定の大きさの領域に区切られる。セクタには，1から始まるセクタ番号が順に付けられている。ファイルは，1個以上のセクタに分割して格納される。

(2) FATは，セクタの状態を管理するための整数型の1次元配列であり，セクタの状態に応じて，表1に示す値が格納されている。

表1　FATの*i*番目の要素の値

*i*番目のセクタの状態		値
未使用		0
使用中	ファイルの末尾以外	次のセクタ番号
	ファイルの末尾	−1

(3) DCT は，ディレクトリ構造及びファイルに関する情報を管理するレコードの配列である。DCT のレコードは，表 2 に示す 6 個の要素からなり，一つのディレクトリ又はファイルに対して 1 個のレコードが作られる。

表 2　DCT のレコード構成

レコード要素	レコード要素の説明
名前	ディレクトリ又はファイルの名前
種別	“ディレクトリ”，“ファイル”，“未使用”のいずれか
上位レコード	所属するディレクトリに対応する DCT のレコード番号 （0 の場合は，ルートディレクトリに所属することを示す）
セクタ数	ディレクトリの場合：　0 ファイルの場合：　ファイルを格納しているセクタ数
作成日時	ディレクトリ又はファイルの作成日時
先頭セクタ	ディレクトリの場合：　“未使用” ファイルの場合：　先頭のセクタ番号

(4) 図 1 の例の場合における DCT と FAT の関係を図 2 に示す。ファイルを格納しているセクタ数は，file1 が 2（セクタ番号 1，2），file2 が 4（セクタ番号 3，6，7，8），file3 が 2（セクタ番号 4，5）である。

DCT

レコード番号	名前	種別	上位レコード	セクタ数	作成日時	先頭セクタ
1	dir1	ディレクトリ	0	0	2004-10-02 09:00	未使用
2	file1	ファイル	1	2	2004-10-02 09:15	1
3	file2	ファイル	1	4	2004-10-02 09:20	3
4	dir2	ディレクトリ	0	0	2004-10-02 10:00	未使用
5	file3	ファイル	4	2	2004-10-02 10:15	4
6		未使用				
7		未使用				
⋮	⋮	⋮	⋮	⋮	⋮	⋮

セクタ番号	FAT
1	2
2	−1
3	6
4	5
5	−1
6	7
7	8
8	−1
9	0
10	0
⋮	

図 2　DCT と FAT の関係の例

(5) ファイル及びディレクトリの生成と削除操作の内部処理は，表3のとおりである。ただし，同一ディレクトリの直下には，同じ名前のファイル又はディレクトリは作成できない。

表3　ファイル及びディレクトリの生成と削除

操作	内部処理の説明
ファイルの生成	① FAT 内を先頭から順番に検索し，ファイルを格納することができるだけの未使用セクタを確保して，内容を書き込むとともに，FAT の内容を更新する。 ② DCT を先頭から順番に検索し，種別が“未使用”であるレコードを一つ確保し，その各要素に値を書き込む。 ③ ②のレコード内の先頭セクタに値を設定する。
ファイルの削除	① DCT の当該レコードの種別を“未使用”にする。 ② ファイルが使用していたセクタに対応する FAT の要素の値を 0 にする。
ディレクトリの生成	DCT を先頭から順番に検索し，種別が“未使用”であるレコードを一つ確保し，その各要素に値を書き込む。
ディレクトリの削除	① DCT の当該レコードの種別を“未使用”にする。 ② このディレクトリに属しているすべてのファイルとディレクトリを削除する。

設問1　図2の状態から，file1を削除し，セクタ数が3のファイルfile4をディレクトリdir2の直下に生成した。file4を生成した直後のFATの要素（セクタ番号1～10）として，正しいものを解答群の中から選べ。

解答群

	1	2	3	4	5	6	7	8	9	10	セクタ番号
ア	2	3	−1	5	−1	7	8	−1	0	0	
イ	2	3	−1	5	−1	7	8	9	−1	0	
ウ	2	6	6	5	−1	−1	8	−1	0	0	
エ	2	6	7	5	6	−1	−1	9	−1	0	
オ	2	9	6	5	−1	7	8	−1	−1	0	
カ	2	9	6	5	−1	7	8	−1	10	−1	

設問2　ファイルの操作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。

あるディレクトリに属するファイルを異なるディレクトリに移動するとき，移動先のディレクトリ直下に［ a ］が存在しない場合には，［ b ］に対応するDCTの当該レコードの上位レコードを移動先のディレクトリに変更する。

移動先のディレクトリ直下に［ a ］が存在する場合には，上書きを行う。そのため，移動先のディレクトリ直下にある［ c ］を行った後，［ b ］に対応するDCTの当該レコードの上位レコードを移動先のディレクトリに変更する。

解答群

ア　移動先のディレクトリ　　イ　移動先のファイル

ウ　移動元のディレクトリ　　エ　移動元のファイル

オ　同じ名前のファイル　　カ　ディレクトリの削除

キ　ディレクトリの生成　　ク　ファイルの削除

ケ　ファイルの生成

問2　ネットワークシステムに関する次の記述を読んで，設問1～3に答えよ。

X社では，社外とのネットワーク接続にインターネットを利用している。社内の各部門ごとにLANがあり，複数のパソコンやサーバを接続する。パソコンやサーバには固定のプライベートIPアドレスを付与する。各部門のLANは，ルータを介して基幹LANに接続する。インターネットへの接続は，基幹LAN上の関門サーバが行う。関門サーバには，プロキシの機能，コンピュータウイルスを検出する機能，ファイアウォールの機能などが含まれている。

X社のネットワーク構成を，Y部門とZ部門を例にとって図に示す。図中の数字は，IPアドレスを表す。

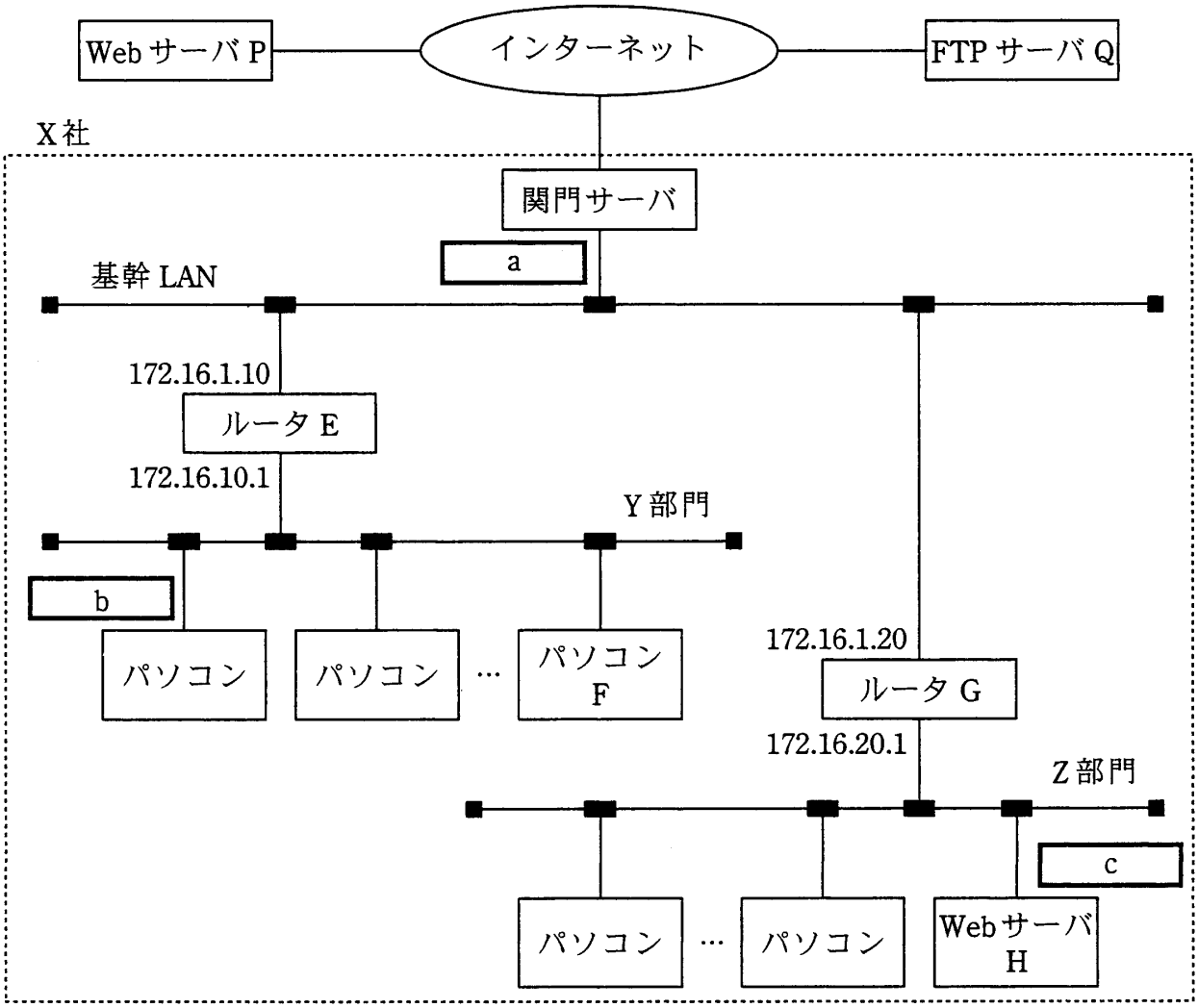


図　X社のネットワーク構成

設問1　図中の［　　］は，パソコンやサーバのIPアドレスを示す。指定可能なIPアドレスを，解答群の中から選べ。ここで，サブネットマスクは，255.255.255.0とする。

解答群

ア	172.16.0.1	イ	172.16.0.2	ウ	172.16.1.1	エ	172.16.2.2
オ	172.16.10.1	カ	172.16.10.2	キ	172.16.10.255		
ク	172.16.20.1	ケ	172.16.20.2	コ	172.16.20.255		

設問2　次の表は，パソコンFのブラウザからZ部門のWebサーバHにデータを送信する場合と，同じブラウザからインターネット上のWebサーバPにデータを送信する場合の，データに付加するIPアドレスやMACアドレスがどのコンピュータのものであるかを示したものである。表中の［　　］に入れる正しい答えを，解答群の中から選べ。ただし，社内のWebサーバにアクセスする場合には，プロキシの機能を使用しないものとする。

表　データに付加するアドレス

送信先	パソコンFが送信するデータ		送信先が受信するデータ
	送信先IPアドレス	送信先MACアドレス	送信元IPアドレス
Webサーバ H	Webサーバ H		
Webサーバ P	Webサーバ P		

解答群

ア

Web サーバ H	パソコン F
Web サーバ P	関門サーバ

イ

Web サーバ H	パソコン F
Web サーバ P	パソコン F

ウ

Web サーバ H	パソコン F
関門サーバ	パソコン F

エ

ルータ E	パソコン F
関門サーバ	パソコン F

オ

ルータ E	パソコン F
ルータ E	関門サーバ

設問 3　次の記述中の［　　］に入れる正しい答えを，解答群の中から選べ。

パソコン F からインターネット上の FTP サーバ Q に FTP クライアントソフトで接続を試みたところ，うまく接続できなかった。しかし，パソコン F で，FTP サーバ Q を指定した ping コマンドを実行したところ，FTP サーバ Q から応答があった。接続のための操作手順が正しいとすると，関門サーバの［　　］の設定に原因がある場合がある。

解答群

ア　Java アプレットの使用許可
イ　クッキーの受入れ
ウ　サブネットマスク
エ　通過可能なポート番号
オ　ルーティングテーブル

問3　コンピュータ資源へのアクセス権の設定に関する次の記述を読んで，設問に答えよ。

あるプログラム実行環境では，プログラムが実行時に使用できるコンピュータ資源（以下，資源と呼ぶ）へのアクセス権をテキストファイルに記述する。プログラム実行環境は，プログラムの実行時に，資源へのアクセス要求があるごとに記述されたアクセス権を評価し，要求された資源へのアクセスを許可するかどうかを決める。テキストファイルはgrant文の並びであり，grant文の形式は次のとおりである。

```
grant　プログラム名　資源の種類　資源名　アクセス権 ;
```

(1) grant文はキーワード`grant`で始まり`;`（セミコロン）で終わる。

(2) プログラム名，ファイル名及びホスト名は`"`（二重引用符）で囲み，`*`をワイルドカード文字として使用できる。例えば，`"a*n"`はaで始まりnで終わる任意の長さの文字列を表し，`"an"`，`"afternoon"`などと一致する。

(3) 資源は，資源の種類と資源名で指定する。指定できる資源の種類及び資源名は，次のとおりである。

区分	資源の種類	資源名
ローカルファイル	`file`	ファイル名を`"`で囲む。複数あるときは`"file A","file B"`のようにコンマで区切る。
ネットワーク	`network`	ホスト名を`"`で囲む。複数あるときは`"example.jp","example.ne.jp"`のようにコンマで区切る。
入力イベント(注)	`inputevent`	`"keyboard"`でキーボードからの入力イベントを，`"mouse"`でマウスからの入力イベントを表す。`"keyboard","mouse"`で両方を表す。

注　プログラム実行環境では，入力装置から発生する入力イベントを資源とみなす。

(4) アクセス権は，資源の種類ごとに次のとおり指定する。アクセス権はキーワードで示し，複数あるときはコンマで区切る。

資源の種類	アクセス権	
	キーワード	意味
file	read	ファイルの読出しを許可する。
	write	ファイルの作成，削除及びファイルへの書込みを許可する。
network	accept	指定されたホストからのネットワーク接続を許可する。
	connect	指定されたホストへのネットワーク接続を許可する。
inputevent	create	入力イベントの生成を許可する。
	listen	入力イベントの受取りを許可する。

例えば，texteditor というプログラムにすべてのテキストファイル（ファイル名は .txt で終わるものとする）の読み書きとキーボードからの入力イベントの受取りを許可するとき，次のとおりに記述する。

```
grant "texteditor" file "*.txt" read,write;
grant "texteditor" inputevent "keyboard" listen;
```

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。

あるサイトからダウンロードしたプログラムを実行する場合を想定する。ダウンロードしたプログラムが信頼できるかどうかわからないときに，そのプログラムを試用するために必要な最小限のアクセス権の設定を行いたい。例えば，GUI（グラフィカルユーザインタフェース）とゲームサーバ用のホスト games.example.jp を使用し，ネットワークを通して行う対戦型ゲームプログラム netgame を考える。GUI 操作に必要なキーボードとマウスからの入力イベントの受取り及び netgame からホスト games.example.jp へのネットワーク接続に限って許可するとき，次のとおりに記述する。

```
grant "netgame" inputevent "keyboard","mouse" [ a ];
grant "netgame" network "games.example.jp" [ b ];
```

同様に，かな漢字変換入力プログラム kkinput とその辞書 kkdict.dat をダウンロードしたとする。kkinput は，入力イベントフィルタとして働き，キーボードから入力イベントとして受け取った平仮名やローマ字を，辞書を参照して漢字に変換し，キーボードからの入力イベントを破棄し，新たにキーボードの入力イベントを生成してアプリケーションに通知する。学習及び単語登録用の個人辞書 mydict.dat が使用できる。実行に必要な最小限のアクセス権を設定してこのプログラムを試用したいときは，次のとおりに記述する。ただし，kkinput は netgame と組み合わせて使用しないものとする。

```
grant "kkinput" inputevent "keyboard" [ c ];
grant "kkinput" file [ d ];
grant "kkinput" file [ e ];
```

a ～ c に関する解答群

ア accept,connect	イ connect	ウ create
エ create,listen	オ listen	カ read
キ read,write	ク write	

d, e に関する解答群

ア "*.dat" read
イ "*.dat" read,write
ウ "*.dat" write
エ "kkdict.dat" read
オ "kkdict.dat" read,write
カ "kkdict.dat" write
キ "mydict.dat" read
ク "mydict.dat" read,write
ケ "mydict.dat" write

問4　次のプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

関数 RadixConv は，M 進数字列（2 ≦ M ≦ 16）を，N 進数字列（2 ≦ N ≦ 16）に基数変換するプログラムである。

(1) M 進数字列は，文字である M 進数字だけで構成され，空白文字は含まない。11 ～ 16 進数の 10 以上の数字には，“A” ～ “F” のアルファベットを用いる。

(2) RadixConv は，M 進数字列を整数値に変換した後，その整数値を N 進数字列に変換する。M 進数字列を整数値に変換する関数は MToInt，整数値を N 進数字列に変換する関数は IntToN である。

(3) 関数 MToInt と関数 IntToN では，次の関数を利用する。

① 1 文字の数字 P（“0”，“1”，…，“F” のいずれか）を整数値に変換する関数 ToInt

② 整数値 Q（0 ≦ Q ≦ 15）を 1 文字の数字（“0”，“1”，…，“F” のいずれか）に変換する関数 ToStr

③ 文字列の長さを返す既定義の関数 Length

④ 文字列の一部を取り出す既定義の関数 Substr

(4) 関数の引数と返却値の仕様を表 1 ～ 5 に示す。

表 1　RadixConv

引数名／返却値	データ型	意味
Frdx	整数型	変換元の数字列の基数（2 ≦ Frdx ≦ 16）
Fnum	文字型	変換元の数字列
Trdx	整数型	変換後の数字列の基数（2 ≦ Trdx ≦ 16）
返却値	文字型	変換後の Trdx 進数字列

表2　MToInt

引数名／返却値	データ型	意味
Rdx	整数型	変換元の数字列の基数（2 ≦ Rdx ≦ 16）
Num	文字型	変換元の数字列
返却値	整数型	変換後の整数

表3　IntToN

引数名／返却値	データ型	意味
Val	整数型	変換元の整数
Rdx	整数型	変換後の数字列の基数（2 ≦ Rdx ≦ 16）
返却値	文字型	変換後の Rdx 進数字列

表4　ToInt

引数名／返却値	データ型	意味
P	文字型	変換元の 1 文字の数字（“0”, “1”, …, “F”）
返却値	整数型	変換後の整数

表5　ToStr

引数名／返却値	データ型	意味
Q	整数型	変換元の整数（0 ≦ Q ≦ 15）
返却値	文字型	変換後の 1 文字の数字

〔プログラム〕

```
○文字型: RadixConv(整数型: Frdx, 文字型: Fnum, 整数型: Trdx)
・return IntToN(MToInt(Frdx, Fnum), Trdx)
                    /* IntToN の値を関数の返却値とする */

○整数型: MToInt(整数型: Rdx, 文字型: Num)
○整数型: Idx, Val
・Val ← 0
■ Idx: 1, Idx ≦ Length(Num), 1   /* Length は Num の文字列長を返す */
  ・Val ← [    a    ] + ToInt(Substr(Num, Idx, 1))
          /* Substr は Num の先頭から Idx 番目（≧1）の1文字を取り出す */
・return Val          /* Val を関数の返却値とする */

○文字型: IntToN(整数型: Val, 整数型: Rdx)
○整数型: Quo          /* 商 */
○整数型: Rem          /* 剰余 */
○文字型: Tmp
・[    b    ]
・Tmp ← ""
■ Quo ≧ Rdx
  ・Rem ← Quo % Rdx
  ・Tmp ← ToStr(Rem) + Tmp        /* + は文字列を連結する演算子 */
  ・[    c    ]
・[    d    ]
・return Tmp                  /* Tmp を関数の返却値とする */

○整数型: ToInt(文字型: P)
○整数型: Idx
○文字型: Code[16]             /* 添字は0から始まる */
/* Code には，初期値として"0", "1", "2", "3", "4", "5", "6", "7", */
/* "8", "9", "A", "B", "C", "D", "E", "F" がこの順に格納されている */
/* 文字の値もこの順に大きくなる */
・Idx ← 0
■ [    e    ]                  /* 文字の比較 */
  ・Idx ← Idx + 1
・return Idx                  /* Idx を関数の返却値とする */

○文字型: ToStr(整数型: Q)
○文字型: Code[16]             /* 添字は0から始まる */
/* Code には，初期値として"0", "1", "2", "3", "4", "5", "6", "7", */
/* "8", "9", "A", "B", "C", "D", "E", "F" がこの順に格納されている */
/* 文字の値もこの順に大きくなる */
・return Code[Q]          /* Code[Q] を関数の返却値とする */
```

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　Rdx　　イ　Val　　ウ　Val × Rdx

エ　Val ÷ Rdx

b ～ d に関する解答群

ア　Quo ← Quo ÷ Rdx　　イ　Quo ← Quo ÷ Rem

ウ　Quo ← Rdx　　エ　Quo ← Rem ÷ Rdx

オ　Quo ← Val　　カ　Rem ← Rdx

キ　Rem ← Val　　ク　Tmp ← ToStr(Quo) + Tmp

ケ　Tmp ← ToStr(Rem) + Tmp

e に関する解答群

ア　P ＜ Code[Idx]　　イ　P ＞ Code[Idx]

ウ　P ≦ Code[Idx]　　エ　P ≧ Code[Idx]

問5　プログラム設計に関する次の記述を読んで，設問1，2に答えよ。

ある小売業者は，本社と30の支店からなっている。毎日各支店から本社に次に示すレコード様式の各商品の売上実績ファイルが送付される。

売上実績ファイルのレコード様式

年月日	支店コード	商品コード	販売数量	売上金額

本社でまとめられた売上実績ファイルは，支店コード別，商品コード別で昇順に整列されている。扱う商品の種類の総数は8,000種類であり，各支店で取扱いのない商品及び売上げがない商品も売上金額が0として記録されている。各支店では平均6,000種類の商品が取り扱われている。

本社では，売上実績ファイルから支店別の売上実績表を作成しており，全支店での売上金額の多い順に50商品を選び，それらの商品が各支店では何番目の売上げであったかを，全支店での順位とともに図1に示すように印字する。売上金額が等しい商品には，商品コードの昇順に順位を付ける。

なお，この問題において整列処理は安定であるとする。

支店コード	0023						
商品コード	商品名称	支店データ			全支店データ		
		順位	販売数量	売上金額	順位	販売数量	売上金額
002411	芳香剤	298	8	3,360	1	3,112	1,307,040
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
002245	シャンプー	14	68	20,264	50	1,472	438,656

図1　全支店での売上順位を基にした売上実績表

支店別の売上実績表を作成する方法について，図2に示す処理の流れ図を作成した。

〔売上上位50商品の抽出〕

商品コードをキーとして売上実績ファイルを整列する。商品コードごとの販売数量の合計と売上金額の合計とを計算し，売上金額の多い順に上位50商品をFファイル

に出力する。

〔支店ごとの順位付け〕

支店コード，売上金額（降順）をキーとして売上実績ファイルを整列する。支店ごとに売上金額の多い順に支店データの順位を付け，Gファイルに出力する。

〔全支店データと支店別データの照合〕

全支店データ（Fファイル）をメモリに格納した後，支店別データ（Gファイル）と照合して，1レコードに編集した次の様式のHファイルを作成する。

Hファイルのレコード様式

支店コード	商品コード	支店データ			全支店データ		
		順位	販売数量	売上金額	順位	販売数量	売上金額

〔支店別売上実績表の作成〕

支店コード，全支店データの順位をキーとしてHファイルを整列する。商品名称表示のために商品マスタファイルを参照しながら，支店別に売上実績表を印字する。

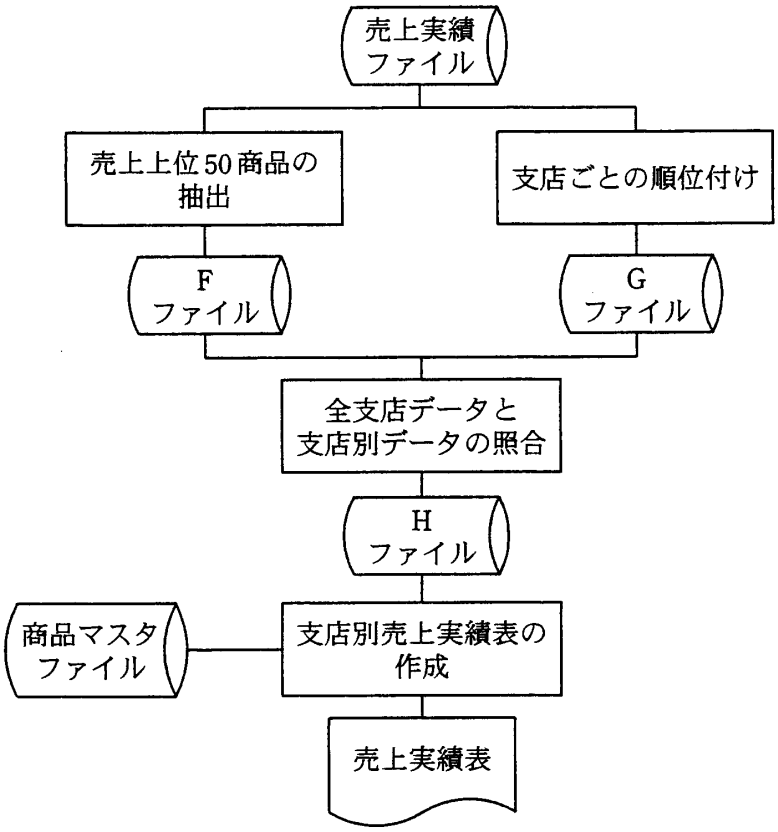


図2　全支店での売上順位を基にした売上実績表の作成処理

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。

図2の処理で，Fファイルとして必要最小限の項目をもつレコード様式は［　a　］である。また，Hファイルのレコード件数は［　b　］件である。

aに関する解答群

ア

年月日	支店コード	商品コード	販売数量	売上金額

イ

年月日	販売数量合計	売上金額合計

ウ

支店コード	商品コード	販売数量合計	売上金額合計

エ

支店コード	販売数量合計	売上金額合計

オ

商品コード	販売数量合計	売上金額合計

bに関する解答群

ア　1,500	イ　6,000	ウ　8,000	エ　40,000
オ　180,000	カ　240,000	キ　320,000	ク　400,000

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。

各支店で売上金額の多い順に50商品を選び，それらの商品が全支店では何位の売上げであったかを，支店データの順位とともに図3に示すように印字する。

なお，売上金額が等しい商品は，商品コードの昇順に順位を付ける。

支店コード	0023						
商品コード	商品名称	支店データ			全支店データ		
		順位	販売数量	売上金額	順位	販売数量	売上金額
001203	石けん	1	314	30,772	523	510	49,980
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
010204	歯ブラシ	50	221	15,028	238	1,477	100,436

図3　支店ごとの売上順位を基にした売上実績表

この売上実績表を作成するために，図4に示す処理の流れ図を作成した。

〔売上合計の計算〕

商品コードをキーとして売上実績ファイルを整列する。商品コードごとの販売数量の合計と売上金額の合計とを計算し，全件をLファイルに出力する。

〔ステップ1〕

Lファイルを［　c　］に整列し，全支店順位を付けて，全件をL1ファイルに出力する。

〔ステップ2〕

L1ファイルを［　d　］に整列し，L2ファイルを作成する。

〔売上上位商品の抽出〕

支店コード，売上金額（降順）をキーとして売上実績ファイルを整列する。支店ごとに商品の売上金額による順位を付け，売上実績表の作成に必要な［　e　］件だけをMファイルに出力する。

〔商品コードで整列〕

Mファイルを商品コードの昇順に整列し，M1ファイルを作成する。

〔全支店データと支店別データの照合〕

L2ファイル（全支店データ）とM1ファイル（支店別データ）を照合して，Nファイルを作成する。

〔支店別売上実績表の作成〕

支店コード，その支店データの順位をキーとしてNファイルを整列する。商品名称表示のために商品マスタファイルを参照しながら，支店別に売上実績表を印字する。

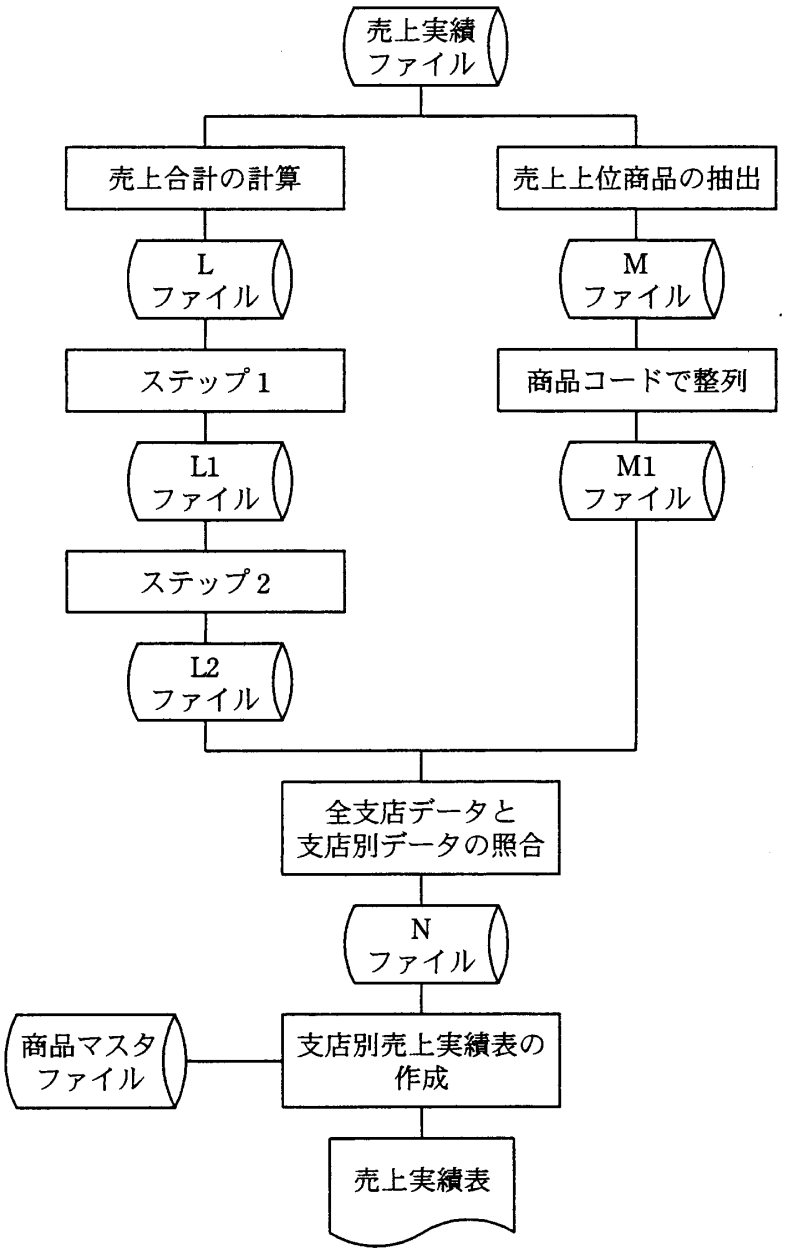


図4　支店ごとの売上順位を基にした売上実績表の作成処理

c, d に関する解答群

ア　売上金額の昇順	イ　売上金額の降順
ウ　商品コードの昇順	エ　商品コードの降順
オ　全支店データの順位の昇順	カ　全支店データの順位の降順

e に関する解答群

ア　1,500	イ　6,000	ウ　8,000	エ　40,000
オ　180,000	カ　240,000	キ　320,000	ク　400,000

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〕

CGI などへのリクエストの際に文字列パラメタを URL に含める場合には，その文字列パラメタを決められた規則に従って変換し送出する必要がある。このプログラムは，それに用いる変換プログラム `URLEncode` である。

(1) 変換対象の文字列パラメタは，JIS X 0201（7 ビット及び 8 ビットの情報交換用符号化文字集合）の文字で構成される。また，文字列パラメタの途中にナル文字（文字符号 0x00）は含まれないものとする。

(2) 変換規則は次のとおりである。

① 英数文字（0x30 ～ 0x39，0x41 ～ 0x5A，0x61 ～ 0x7A）及び“@”（0x40），“*”（0x2A），“-”（0x2D），“.”（0x2E），“_”（0x5F）は変換しない（これらの文字を以降，無変換文字と呼ぶ）。

② ①に含まれない文字は，“%”の後に文字符号の 2 けたの 16 進表記を続けた 3 文字に変換する。例えば，文字符号 0x5E の文字は“%5E”と変換する。

(3) 関数 `URLEncode` の仕様は，次のとおりである。

形式：
```
void URLEncode( unsigned char *input,
                unsigned char *output )
```

引数：　`input`　変換前の文字列が格納されている文字型配列へのポインタ

`output`　変換後の文字列を格納する文字型配列へのポインタ

(4) `output` が指す配列は，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。

C

(5) 例えば，“Hi!”という文字列パラメタを変換した結果は図のとおりとなる。

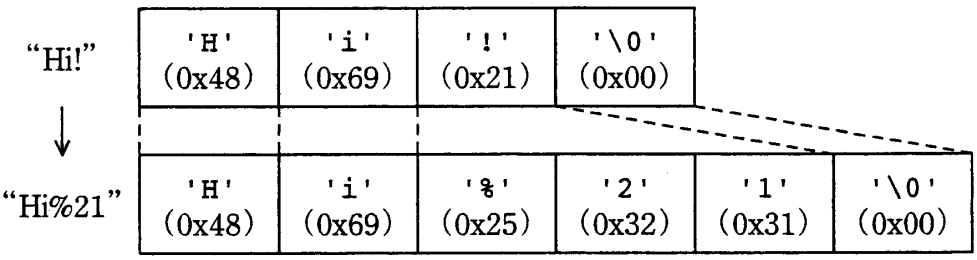


図　文字列パラメタの変換例

C

(6) 次の関数があらかじめ用意されている。

形式：　`int replaceChar( unsigned char c )`

引数：　`c`

返却値：文字 c が無変換文字の場合は 0 を，それ以外の場合は 1 を返す。

〔プログラム〕

```
int replaceChar( unsigned char );

void URLEncode( unsigned char *input, unsigned char *output ) {

    const unsigned char chars[] = "0123456789ABCDEF";

    while( *input != '\0' ) {
        if ( replaceChar( [ a ] ) ) {
            *output++ = '%';
            *output++ = chars[ [ b ] ];
            *output++ = chars[ [ c ] ];
        } else {
            *output++ = *input;
        }
        [ d ];
    }
    *output = '\0';
}
```

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, d に関する解答群

ア　`&input`　　イ　`(&input)++`　　ウ　`(*input)++`

エ　`**input`　　オ　`**input++`　　カ　`*input`

キ　`input`　　ク　`input++`

b, c に関する解答群

ア　`*input << 4`　　イ　`*input >> 4`

ウ　`*input & 0x0F`　　エ　`*input & 0xF0`

オ　`*input ^ 0x0F`　　カ　`*input ^ 0xF0`

キ　`*input | 0x0F`　　ク　`*input | 0xF0`

C

問7　次のCOBOLプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

研修申込状況を登録した申込ファイルから，研修番号別に申込者の人数を集計して印字する。

(1) 申込ファイルのレコード様式は，次のとおりである。

申込番号 5けた	研修番号		その他 42けた
	内容区分 1けた	開催区分 2けた	

① 研修を申し込むと申込者1人について一つの申込番号が発行され，1件のレコードが申込ファイルに登録される。

② 同一人が，同じ研修に二度以上申し込むことはない。

③ 研修番号は，内容区分と開催区分から構成されている。

④ 内容区分は，A～Hのいずれか1文字である。開催区分は，01～20のいずれかである。

⑤ 申込ファイルの内容には，間違いがないものとする。

(2) 集計表の印字様式は，次のとおりである。

ｹﾝｼｭｳﾊﾞﾝｺﾞｳ	ﾓｳｼｺﾐｼｬｽｳ
XXX	ZZ9
XXX	ZZ9
XXX	ZZ9
⋮	⋮
XXX	ZZ9

① 各研修とも申込者数は，1,000未満とする。

② 見出しは，最初に1回だけ印字する。

③ 申込者数が0の研修は，印字しない。

〔プログラム〕

```
DATA DIVISION.
FILE SECTION.
FD  MOSHIKOMI-FILE.
01  MOSHIKOMI-REC.
    02                  PIC  X(05).
    02  KENSHU-M.
        03  KEN-N-M     PIC  X(01).
        03  KEN-K-M     PIC  9(02).
    02                  PIC  X(42).
FD  SHUKEI-PRINT.
01  SHUKEI-REC.
    02  KENSHU-S.
        03  KEN-N-S     PIC  X(01).
        03  KEN-K-S     PIC  9(02).
    02                  PIC  X(15).
    02  SU-S            PIC  ZZ9.
WORKING-STORAGE SECTION.
01  FILE-END            PIC  X(01) VALUE "N".
01  LOOP1               PIC  9(01).
01  LOOP2               PIC  9(02).
01  KEN-NO              PIC  9(01).
01  KEN-W1.
    02  KEN-W2          PIC  X(08) VALUE "ABCDEFGH".
    02  REDEFINES KEN-W2.
        03 KEN-W3       PIC  X(01) OCCURS 8 INDEXED BY KEN.
01  SU-WK1                        VALUE ZERO.
    02  OCCURS 8.
        03  SU-WK       PIC  9(03) OCCURS 20.
01  MIDASHI             PIC  X(21) VALUE "ｹﾝｼｭｳﾊﾞﾝｺﾞｳ ﾓｳｼｺﾐｼｬｽｳ".
PROCEDURE DIVISION.
MAIN-RTN.
    OPEN INPUT MOSHIKOMI-FILE OUTPUT SHUKEI-PRINT.
    PERFORM KASAN-RTN UNTIL FILE-END = "Y".
    PERFORM PRINT-RTN.
    CLOSE MOSHIKOMI-FILE SHUKEI-PRINT.
    STOP RUN.
KASAN-RTN.
    READ MOSHIKOMI-FILE
       AT END
          MOVE "Y" TO FILE-END
       NOT AT END
          [ a ]
          SEARCH [ b ]
             WHEN KEN-N-M = KEN-W3 (KEN)
                SET KEN-NO TO KEN
          END-SEARCH
          ADD 1 TO SU-WK (KEN-NO KEN-K-M)
    END-READ.
```

COBOL

```
PRINT-RTN.
    WRITE SHUKEI-REC FROM MIDASHI AFTER PAGE.
    PERFORM VARYING LOOP1 FROM 1 BY 1 UNTIL LOOP1 > 8
       PERFORM VARYING LOOP2 FROM 1 BY 1 UNTIL LOOP2 > 20
          MOVE SPACE TO SHUKEI-REC
          IF [          c          ] NOT = 0 THEN
             MOVE KEN-W3 (LOOP1) TO KEN-N-S
             MOVE LOOP2 TO KEN-K-S
             MOVE [          c          ] TO SU-S
             WRITE SHUKEI-REC AFTER 1
          END-IF
       END-PERFORM
    END-PERFORM.
```

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　MOVE 1 TO KEN　　イ　MOVE 1 TO KEN-NO

ウ　MOVE KEN TO KEN-NO　　エ　MOVE KEN-NO TO KEN

オ　SET KEN TO 1　　カ　SET KEN TO KEN-N-S

キ　SET KEN TO KEN-NO　　ク　SET KEN-NO TO 1

ケ　SET KEN-NO TO KEN

bに関する解答群

ア　KEN　　イ　KEN-NO　　ウ　KEN-W1

エ　KEN-W2　　オ　KEN-W3　　カ　SU-WK

cに関する解答群

ア　SU-WK (KEN-K-M KEN-NO)　　イ　SU-WK (KEN-K-S KEN-NO)

ウ　SU-WK (KEN-NO KEN-K-M)　　エ　SU-WK (KEN-NO KEN-K-S)

オ　SU-WK (LOOP1 LOOP2)　　カ　SU-WK (LOOP2 LOOP1)

問8　次のJavaプログラムの説明及びプログラムを読んで，設問に答えよ。

〔プログラムの説明〕

任意のオブジェクトを格納し，取り出すプログラムである。格納されたオブジェクトを取り出す方法として，先入れ先出し法（First-In-First-Out）及び後入れ先出し法（Last-In-First-Out）がある。

(1) 抽象クラス `Store` は，メソッド `put`，`get`，`size` を定義する。

`public void put(Object value)`

引数で指定した `value` を `Store` のインスタンスに先着順に格納する。格納できるオブジェクトの個数の上限は50である。それを超えて格納しようとしたときは `DataStoreException` を投げる。

`public abstract Object get()`

格納されているオブジェクトを一つ取り出して返す。どのオブジェクトを取り出すかは，このメソッドを実装するサブクラスによって決まる。オブジェクトがないときは `DataStoreException` を投げる。

`public int size()`

格納されているオブジェクトの個数を返す。オブジェクトが格納されていないときは0を返す。

(2) クラス `FifoStore` は，抽象クラス `Store` のサブクラスで，メソッド `get` は格納されているオブジェクトのうち最初に格納されたものを取り出して返す。すなわち，先入れ先出しとなる。

(3) クラス `LifoStore` は，抽象クラス `Store` のサブクラスで，メソッド `get` は格納されているオブジェクトのうち最後に格納されたものを取り出して返す。すなわち，後入れ先出しとなる。

(4) クラス `StoreTest` は二つのサブクラスをテストするプログラムである。プログラム起動時に指定された引数を `FifoStore` 及び `LifoStore` に格納し，取り出す操作をする。実行例を図に示す。ただし，図中の `%` はシステムのコマンドプロンプトを表し，コマンドの引数は `String` の配列としてメソッド `main` に渡されるものとする。

```
% java StoreTest 起 承 転 結
0: 起 1: 承 2: 転 3: 結
0: 結 1: 転 2: 承 3: 起
```

図　クラス StoreTest の実行例

〔プログラム 1〕

```
public abstract class Store {
   Object[] data = new Object[50];
   int index = 0;

   public void put(Object value) {
      if ( [ a ] )
         throw new DataStoreException("overflow");
      data[index++] = value;
   }
   public abstract Object get();
   public int size() {
      return [ b ];
   }
}
```

〔プログラム 2〕

```
public class FifoStore extends Store {
   public Object get() {
      if (index == 0)
         throw new DataStoreException("not exist");
      Object value = data[0];
      for ( [ c ] )
         data[i] = data[i + 1];
      data[--index] = null;
      return value;
   }
}
```

〔プログラム3〕

```
public class LifoStore extends Store {
   public Object get() {
      if (index == 0)
         throw new DataStoreException("not exist");
      Object value = data[--index];
      data[index] = null;
      return value;
   }
}
```

〔プログラム4〕

```
public class StoreTest {
   public static void main(String[] args) {
      FifoStore fifo = new FifoStore();
      LifoStore lifo = new LifoStore();
      for (int i = 0; i < args.length; i++) {
         fifo.put(args[i]);
         lifo.put(args[i]);
      }
      printData(fifo);
      printData(lifo);
   }
   private static void printData( [ d ] store) {
      int size = store.size();
      for (int i = 0; i < size; i++)
         System.out.print(i + ": " + store.get() + " ");
      System.out.println();
   }
}
```

〔プログラム5〕

```
public class DataStoreException extends RuntimeException {
   public DataStoreException(String message) {
      super(message);
   }
}
```

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　index < data.length　　イ　index <= data.length

ウ　index > data.length　　エ　index >= data.length

オ　index + 1 >= data.length

b に関する解答群

ア　data.length　　イ　data.length - 1

ウ　data.length - index　　エ　index

オ　index + 1

c に関する解答群

ア　int i = 0; i < index; i++

イ　int i = 0; i < index - 1; i++

ウ　int i = 0; i < index - 2; i++

エ　int i = 1; i < index; i++

オ　int i = 1; i < index - 1; i++

d に関する解答群

ア　FifoStore　　イ　LifoStore　　ウ　Object

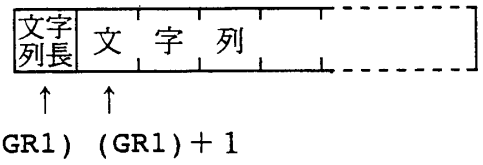
エ　Store　　オ　StoreTest

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，設問に答えよ。

〔プログラムの説明〕

文字列中の二つ以上連続する間隔文字（以下，空白という）を一つに詰める副プログラム SPCSUP である。

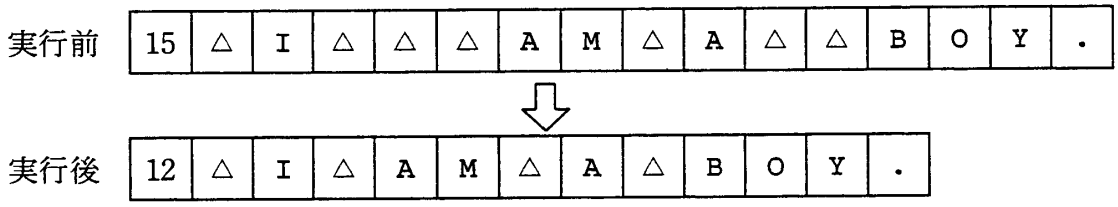
(1) 主プログラムは，先頭に文字列長，その後に文字列が格納された領域の先頭アドレスを GR1 に設定して，副プログラム SPCSUP を呼ぶ。



（文字列長＞0）

(2) 副プログラム SPCSUP は，文字列中の空白を詰めた後，文字列長を再設定する。

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



注　△は空白を示す。

〔プログラム〕

```
SPCSUP  START
        RPUSH
        LD      GR2,GR1          ; 転送元ポインタ(GR2)初期化
        LD      GR3,GR1          ; 転送先ポインタ(GR3)初期化
        LD      GR4,GR1          ;
        +-------------------+    ;  } 文字列終了位置ポインタ(GR4)初期化
        |         a         |    ;
        +-------------------+
        LD      GR6,=0           ; 連続空白識別フラグ(GR6)初期化
LP      LAD     GR2,1,GR2
        LD      GR5,0,GR2
        CPL     GR5,=' '
        +-------------------+
        |         b         |
        +-------------------+
        LD      GR6,GR6
        JNZ     CONT
        +-------------------+
        |         c         |
        +-------------------+
        JUMP    CHMOVE
```

```
RESET   LD      GR6,=0
CHMOVE  LAD     GR3,1,GR3
        ST      GR5,0,GR3
CONT    CPL     GR2,GR4
        JMI     LP
        SUBL    GR3,GR1          ; 新文字列長の算出
        ST      GR3,0,GR1
        RPOP
        RET
        END
```

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　ADDL　GR3,0,GR1　　イ　ADDL　GR4,0,GR1

ウ　LAD　GR1,1,GR1　　エ　LAD　GR3,1,GR3

オ　LAD　GR4,1,GR4

b に関する解答群

ア　JNZ　CHMOVE　　イ　JNZ　CONT　　ウ　JNZ　RESET

エ　JZE　CHMOVE　　オ　JZE　CONT　　カ　JZE　RESET

c に関する解答群

ア　LAD　GR2,1,GR2　　イ　LAD　GR3,1,GR3

ウ　LD　GR6,1　　エ　LD　GR6,=1

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，設問に答えよ。

〔プログラムの説明〕

画面上に表示されるマーカを操作することによって，線画を描く関数 execute である。

(1) 水平方向に800画素，垂直方向に600画素からなるビットマップ画面がある。画面の座標系を図1に示す。画面上には，位置と進行方向の情報をもつマーカ（図1中の•）が表示されている。マーカの進行方向は，上，下，左，右の4通りのいずれかである。マーカが移動すると，その軌跡が描画される。

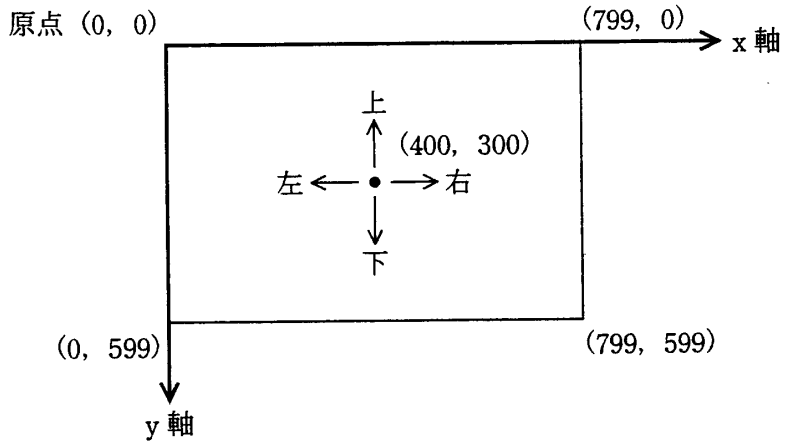


図1　画面の座標系とマーカの初期状態

(2) マーカは MARKER 型の構造体 mark で表現する。プログラムの実行開始時には，マーカの位置は（400, 300），進行方向は上に設定される。

```
typedef struct {
    int x;     /* マーカの x 座標                          */
    int y;     /* マーカの y 座標                          */
    int dir;   /* マーカの進行方向 0:右 1:上 2:左 3:下     */
} MARKER;
MARKER mark  = {400,   /* マーカの初期の x 座標            */
                300,   /* マーカの初期の y 座標            */
                  1    /* マーカの初期の進行方向（上）     */
               };
```

(3) マーカを操作するための命令が定義されている。命令は，命令コードと値から構成され，構造体 INST で表現する。

```
typedef struct {
    char code; /* 命令コード */
    int  val;  /* 値         */
} INST;
```

命令は，構造体 INST の配列である insts の先頭から実行する順番に格納しておく。

(4) 命令コードとその説明を次に示す。

命令コード	説明
{	次の命令から対となる命令コード '}' の直前の命令までを val 回繰り返す。val は 1 以上の整数値である。
t	マーカの進行方向を反時計回りに 90° × val だけ変更する。valは非負の整数値である。
f	現在の進行方向にマーカを val 画素分だけ進め，移動元から移動先まで線分を描く。val は任意の整数値である。
}	繰り返す一連の命令の終了を示す。val は参照しない。
\0	命令列の終了を示す。val は参照しない。

(5) 例えば，図 2 のように構造体配列 insts に命令を格納して，関数 execute を実行した場合の出力は図 3 のとおりである。ただし，図 3 中の座標値は説明のために添えたものであり，実際は出力されない。

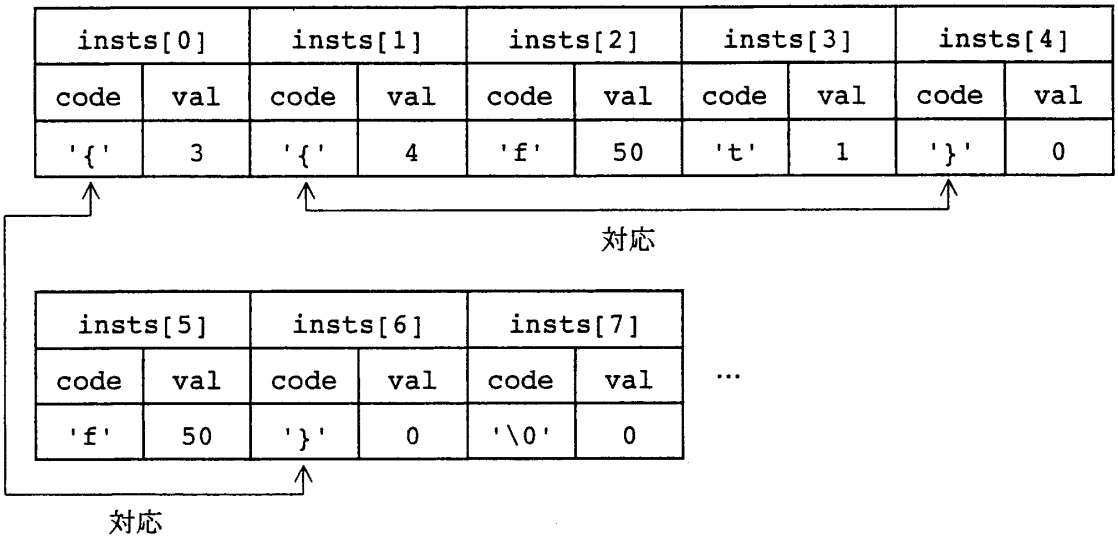


図２　構造体配列 insts に格納されている命令の例

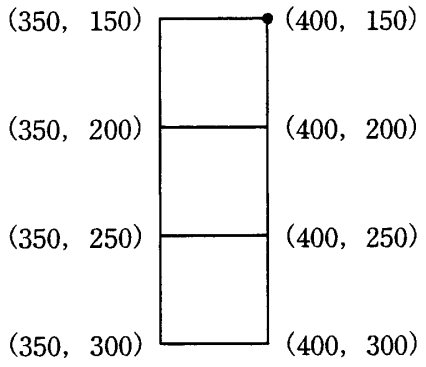


図３　図２の例の実行結果

(6) 線分を描くために次の関数が用意されている。

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

機能：座標（x1, y1）と座標（x2, y2）を結ぶ線分のうち，画面領域に含まれる部分を描画する。

(7) マーカに関する次の関数が用意されている。

```
void eraseMarker( MARKER mark );
```

機能：マーカが画面領域内にあるとき，マーカの表示を消去する。

```
void paintMarker( MARKER mark );
```

機能：マーカが画面領域内にあるとき，マーカを描画する。

(8) マーカが画面領域外に移動し画面上に表示されなくても，その位置座標及び進行方向の情報は保持される。

〔プログラム〕

```
#define INSTSIZE 100  /* 命令数の上限       */
#define STACKSIZE 50  /* {}の入れ子の上限   */

typedef struct {
    int x;     /* マーカの x 座標                          */
    int y;     /* マーカの y 座標                          */
    int dir;   /* マーカの進行方向 0:右 1:上 2:左 3:下     */
} MARKER;

typedef struct {
    char code; /* 命令コード */
    int  val;  /* 値         */
} INST;

typedef struct {
    int opno; /* 繰返しの開始が定義されている配列 insts の要素番号    */
    int rest; /* 繰返しの残り回数                                      */
} STACK;

void drawLine( int, int, int, int );
void eraseMarker( MARKER );
void paintMarker( MARKER );

INST insts[INSTSIZE]; /* 命令を格納するための構造体配列    */
MARKER mark  = {400,  /* マーカの初期の x 座標             */
                300,  /* マーカの初期の y 座標             */
                  1   /* マーカの初期の進行方向（上）      */
               };

void execute(){

    STACK stack[STACKSIZE];
    int opno = 0; /* 実行する命令の配列 insts の要素番号    */
    int spt = -1; /* スタックポインタ                       */
    int dx, dy;

    paintMarker( mark );

    while( insts[opno].code != '\0' ){
        switch( insts[opno].code ){
            case '{':
                  stack[ [ a ] ].opno = opno;
                  stack[spt].rest = insts[opno].val;
                  break;
            case 't':
                  mark.dir = [ b ];
                  break;
```

```
            case 'f':
                  eraseMarker( mark );
                  dx = ( mark.dir % 2 == 0 ) ? [ c ];
                  dy = ( mark.dir % 2 == 0 ) ? [ d ];
                  drawLine( mark.x, mark.y,
                            mark.x + dx, mark.y + dy );
                  mark.x += dx;
                  mark.y += dy;
                  paintMarker( mark );
                  break;
            case '}':
                  if ( stack[spt].rest [ e ] ){
                      opno = stack[spt].opno;
                      stack[spt].rest--;
                  } else {
                      [ f ];
                  }
                  break;
        }
        [ g ];
    }
}
```

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　++spt　　イ　--spt　　ウ　spt

エ　spt++　　オ　spt--

b に関する解答群

ア　(mark.dir + insts[opno].val) % 2

イ　(mark.dir + insts[opno].val) % 3

ウ　(mark.dir + insts[opno].val) % 4

エ　mark.dir + insts[opno].val

オ　mark.dir + insts[opno].val % 2

カ　mark.dir + insts[opno].val % 3

キ　mark.dir + insts[opno].val % 4

c, d に関する解答群

ア (1 - mark.dir) * insts[opno].val : 0

イ (2 - mark.dir) * insts[opno].val : 0

ウ (mark.dir - 1) * insts[opno].val : 0

エ (mark.dir - 2) * insts[opno].val : 0

オ 0 : (1 - mark.dir) * insts[opno].val

カ 0 : (2 - mark.dir) * insts[opno].val

キ 0 : (mark.dir - 1) * insts[opno].val

ク 0 : (mark.dir - 2) * insts[opno].val

ケ 0 : mark.dir * insts[opno].val

コ mark.dir * insts[opno].val : 0

e に関する解答群

ア < 0	イ < 1	ウ == 0
エ > 0	オ > 1	

f, g に関する解答群

ア mark.dir++	イ mark.dir--	ウ opno++
エ opno--	オ spt++	カ spt--

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

〔プログラムの説明〕

五つのチャンネルでテレビを視聴できる地域がある。プログラムは，この地域の調査対象世帯がある1日に視聴した番組を記録した視聴データファイルと，同じ日の番組データを記録した番組データファイルを読み込んで，番組ごとに平均視聴率を求めて印字する。プログラムでは，時間を1分の単位に丸めて扱う。例えば，時刻10時00分は，10時00分00秒から10時00分59秒までの時間を丸めたものとして扱う。

(1) 視聴データファイル（VIEW-FILE）は，次のレコード様式の順ファイルである。

チャンネル番号 2けた	検出開始時刻		検出終了時刻	
	時 2けた	分 2けた	時 2けた	分 2けた

① 視聴データファイルには，標本となる調査対象600世帯の1日の視聴データが格納されている。

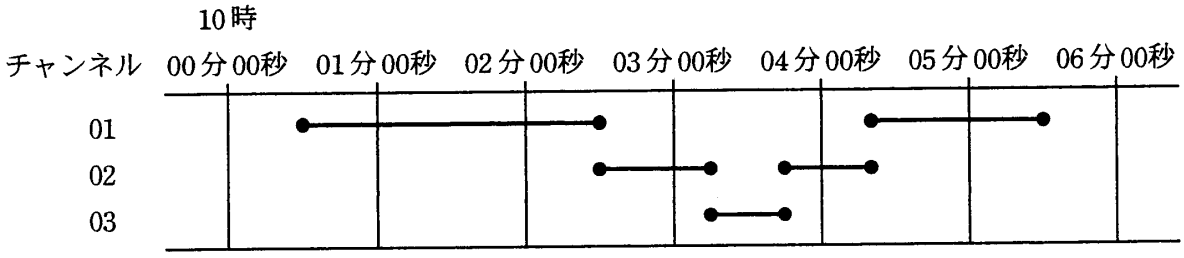
② 1世帯が所有するテレビの台数は1台とする。

③ チャンネル番号は，01～05の五つの値をとる。

④ 一つのレコードは，ある世帯が，“チャンネル番号”のチャンネルを“検出開始時刻”から，“検出終了時刻”まで，視聴したと検出したことを表す。

⑤ 視聴データは，調査対象の世帯が視聴しているチャンネルを，毎分00秒に検出して取得する。

例えば，ある世帯で10時00分30秒から10時05分30秒までテレビを視聴し，その間に次のようにチャンネルを変えたとする。



毎分 00 秒にチャンネルを検出するので，視聴データファイルには次のように分の単位に丸めて記録される。チャンネル 03 の視聴は記録されない。

```
0110011002
0210031004
0110051005
```

⑥ 検出開始時刻及び検出終了時刻の値の範囲は，00 時 00 分 ～ 23 時 59 分である。2 日にわたるデータはない。

⑦ 検出開始時刻 ≦ 検出終了時刻 の関係が成り立つ。

(2) 番組データファイル（PROGRAM-FILE）は，次のレコード様式の順ファイルである。

チャンネル番号 2 けた	番組開始時刻		番組終了時刻		番組名 50 けた
	時 2 けた	分 2 けた	時 2 けた	分 2 けた	

① 番組データファイルには，視聴データファイルと同じ日の，5 チャンネル分の番組データが格納されている。

② 番組開始時刻及び番組終了時刻の値の範囲は，00 時 00 分 ～ 23 時 59 分である。2 日にわたるデータはない。

③ 番組開始時刻 ≦ 番組終了時刻 の関係が成り立つ。

④ 一つのレコードは，ある番組の放送時間が，“番組開始時刻”から“番組終了時刻”までであることを表す。

(3) プログラムでの処理は，次のとおりである。

① 視聴カウント表（VIEW-COUNT-TABLE）は，5（チャンネル）× 1,440（分）の 2 次元の表である。チャンネルごとに，分単位に視聴した世帯数を表の各要素に集計する。

② 番組の平均視聴率は次の式で求める。

$$\text{番組の平均視聴率（\%）} = \frac{\text{各世帯がその番組を視聴した時間の合計}}{\text{標本世帯数} \times \text{その番組の放送時間}} \times 100$$

③ 番組データファイルから読み込んだ番組順に，チャンネル，番組開始時刻，番組終了時刻，平均視聴率（小数第1位までのパーセント表示），番組名を，次の様式で印字する。

```
channel  from - to    rating(%)  program-title
   02    2320 - 2359     8.3      Sports News
   04    0430 - 0439     1.7      Good Morning
   :       :       :      :            :
```

〔プログラム〕

```
DATA DIVISION.
FILE SECTION.
FD  VIEW-FILE.
01  VIEW-REC.
  05  VIEW-CHANNEL              PIC 99.
  05  VIEW-START-HHMM.
    10  VIEW-START-HH           PIC 99.
    10  VIEW-START-MM           PIC 99.
  05  VIEW-END-HHMM.
    10  VIEW-END-HH             PIC 99.
    10  VIEW-END-MM             PIC 99.
FD  PROGRAM-FILE.
01  PROGRAM-REC.
  05  PROG-CHANNEL              PIC 99.
  05  PROG-START-HHMM.
    10  PROG-START-HH           PIC 99.
    10  PROG-START-MM           PIC 99.
  05  PROG-END-HHMM.
    10  PROG-END-HH             PIC 99.
    10  PROG-END-MM             PIC 99.
  05  PROGRAM-TITLE             PIC X(50).
FD  OUT-FILE.
01  OUT-REC                     PIC X(100).
WORKING-STORAGE SECTION.
01  SAMPLE-SIZE                 PIC 9(4)  VALUE 600.
01  M                           PIC 9(4).
01  END-OF-FILE                 PIC X.
01  START-MMMM                  PIC 9(4).
01  END-MMMM                    PIC 9(4).
01  SUMMATION                   PIC 9(9)  BINARY.
01  PROG-RATING                 PIC ZZ9.9.
01  VIEW-COUNT-TABLE.
  05  CHANNEL             OCCURS 5.
    10  COUNT-OF-MINUTE   OCCURS 1440  PIC 9(4)  BINARY.
```

```
01  O-DATA.
  05  FILLER                    PIC X(3) VALUE SPACE.
  05  O-PROG-CHANNEL            PIC 99.
  05  FILLER                    PIC X(3) VALUE SPACE.
  05  O-PROG-START-HHMM.
    10  O-PROG-START-HH         PIC 99.
    10  O-PROG-START-MM         PIC 99.
  05  FILLER                    PIC X(3) VALUE " - ".
  05  O-PROG-END-HHMM.
    10  O-PROG-END-HH           PIC 99.
    10  O-PROG-END-MM           PIC 99.
  05  FILLER                    PIC X(2) VALUE SPACE.
  05  O-PROG-RATING             PIC ZZ9.9.
  05  FILLER                    PIC X(6) VALUE SPACE.
  05  O-PROGRAM-TITLE           PIC X(50).
01  TITLE-LINE                  PIC X(100)
          VALUE "channel from - to    rating(%)  program-title".
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
    PERFORM EXPAND-VIEW-COUNT.
    PERFORM CALCULATE-RATING-AND-PRINT.
    STOP RUN.
EXPAND-VIEW-COUNT.
    [          a          ].
    OPEN INPUT VIEW-FILE.
    MOVE "N" TO END-OF-FILE.
    PERFORM UNTIL END-OF-FILE = "Y"
       READ VIEW-FILE AT END
          MOVE "Y" TO END-OF-FILE
       NOT AT END
          PERFORM SET-VIEW-COUNT
       END-READ
    END-PERFORM.
    CLOSE VIEW-FILE.
CALCULATE-RATING-AND-PRINT.
    OPEN INPUT PROGRAM-FILE  OUTPUT OUT-FILE.
    WRITE OUT-REC FROM TITLE-LINE AFTER ADVANCING 1.
    [          b          ].
    PERFORM UNTIL END-OF-FILE = "Y"
       READ PROGRAM-FILE AT END
          MOVE "Y" TO END-OF-FILE
       NOT AT END
          PERFORM CALCULATE-RATING
          MOVE PROG-CHANNEL    TO O-PROG-CHANNEL
          MOVE PROG-START-HHMM TO O-PROG-START-HHMM
          MOVE PROG-END-HHMM   TO O-PROG-END-HHMM
          MOVE PROG-RATING     TO O-PROG-RATING
          MOVE PROGRAM-TITLE   TO O-PROGRAM-TITLE
          WRITE OUT-REC FROM O-DATA AFTER ADVANCING 1
       END-READ
    END-PERFORM.
    CLOSE PROGRAM-FILE OUT-FILE.
```

```
SET-VIEW-COUNT.
    COMPUTE START-MMMM = VIEW-START-HH * 60 + VIEW-START-MM + 1.
    COMPUTE END-MMMM   = VIEW-END-HH   * 60 + VIEW-END-MM + 1.
    PERFORM VARYING M FROM START-MMMM BY 1 UNTIL M > END-MMMM
        [          c          ]
    END-PERFORM.
CALCULATE-RATING.
    COMPUTE START-MMMM = PROG-START-HH * 60 + PROG-START-MM + 1.
    COMPUTE END-MMMM   = PROG-END-HH   * 60 + PROG-END-MM + 1.
    [          d          ].
    PERFORM VARYING M FROM START-MMMM BY 1 UNTIL M > END-MMMM
        [          e          ]
    END-PERFORM.
    COMPUTE PROG-RATING ROUNDED
      = [          f          ].
```

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, b, d に関する解答群

ア　COMPUTE SUMMATION = END-MMMM - START-MMMM

イ　INITIALIZE VIEW-COUNT-TABLE

ウ　MOVE "N" TO END-OF-FILE

エ　MOVE "Y" TO END-OF-FILE

オ　MOVE SAMPLE-SIZE TO SUMMATION

カ　MOVE ZERO TO SUMMATION

キ　MOVE ZERO TO VIEW-COUNT-TABLE

c, e に関する解答群

ア　ADD 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M)

イ　ADD 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M + 1)

ウ　ADD 1 TO SUMMATION

エ　ADD COUNT-OF-MINUTE(PROG-CHANNEL M) TO SUMMATION

オ　COMPUTE SUMMATION = COUNT-OF-MINUTE(PROG-CHANNEL M + 1)

カ　COMPUTE SUMMATION = COUNT-OF-MINUTE(VIEW-CHANNEL M) + 1

キ　MOVE 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M)

fに関する解答群

ア　100 * SUMMATION / ((END-MMMM - START-MMMM) * (SAMPLE-SIZE + 1))

イ　100 * SUMMATION / ((END-MMMM - START-MMMM + 1) * SAMPLE-SIZE)

ウ　100 * SUMMATION / (END-MMMM - START-MMMM) / SAMPLE-SIZE

エ　100 * SUMMATION / (END-MMMM - START-MMMM + 1) / (SAMPLE-SIZE + 1)

問12　次の Java プログラムの説明及びプログラムを読んで，設問 1，2 に答えよ。

〔プログラムの説明〕

プログラムは，A 社のコールセンタのシミュレータである。A 社はこのシミュレータを，利用者がコールセンタに電話をかけてからオペレータとつながるまでの待ち時間の推定に利用する。コールセンタへの一つの電話呼出しに対しては，1 人のオペレータだけが応答することができる。

このシミュレータでは，空きオペレータがいる限りは利用者を待たせることはないとする。例えば，オペレータが 2 人いるとき，コールセンタに利用者からの電話が 60 秒間隔で 6 回かかってきて，それぞれの通話時間が，130 秒，100 秒，150 秒，90 秒，110 秒，140 秒だったとすると，利用者の待ち時間は 3 番目が 10 秒，5 番目が 30 秒になり，その他は 0 秒である（図 1 参照）。

なお，このシミュレータでは，実世界の 1 秒を 0.1 秒でシミュレートする。

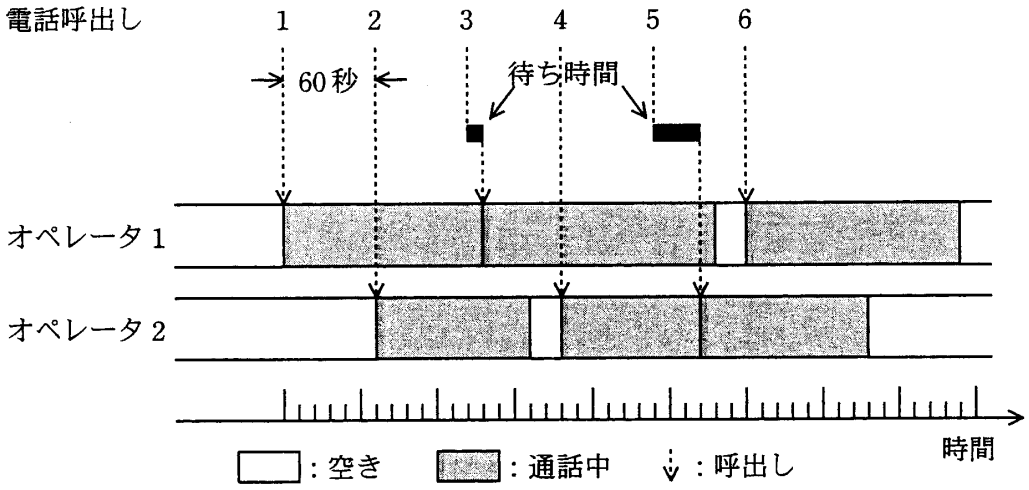


図 1　利用者の待ち時間

プログラムは次のクラスから構成される。

(1) `CallCenter`

コールセンタを表すクラスである。コンストラクタの引数にオペレータ数，通話時間の配列，呼出しを発生させる間隔を指定して，シミュレーションを実行する。次のメソッドと内部クラスをもつ。

`public static void main(String[] args)`

シミュレータをテスト用に起動するメソッドである。

`Call answer()`

オペレータが，利用者からの呼出しに応答するために呼ぶメソッドである。オペレータに割り当てる呼出し（`Call` オブジェクト）を返す。このメソッドは，オペレータに割り当てるべき呼出しが発生するまで，オペレータのスレッドを待たせる。ただし，このクラスが生成するすべての呼出しの割当てが完了し，割り当てるべき呼出しがなくなったときは，`null` を返す。

`Operator`

オペレータをシミュレートするスレッドクラスである。各オペレータの処理は個別のスレッドで実行される。コールセンタにかかってきた電話に応答し，通話する。`CallCenter` が生成するすべての呼出しの割当てが完了するまで，処理を繰り返す。

(2) `Call`

利用者からの呼出しを表すクラスである。コンストラクタの引数には通話時間を指定する。次のメソッドをもつ。

`public void talk()`

オペレータが利用者との通話中の状態をシミュレートするメソッドである。このメソッドは，オペレータに割り当てられた利用者の待ち時間を秒単位（1秒未満は四捨五入）で表示した後，通話時間として指定された時間だけ，オペレータのスレッドを停止させる。

プログラムが図1の例をシミュレートした場合の実行結果を，図2に示す。
なお，図中の `(s)` は（秒）を表す。

```
0(s)
0(s)
10(s)
0(s)
30(s)
0(s)
```

図2　実行結果

プログラム中の java.util.Vector は可変長配列を表すクラスであり，次のメソッドをもつ。

public boolean add(Object obj)

配列の最後尾に要素 obj を追加する。

public Object remove(int index)

index で指定された位置の要素を配列から削除し，削除した要素を返す。先頭の要素の位置は 0 である。index より後方の要素は前方に詰められる。

public boolean isEmpty()

要素がなければ true を，あれば false を返す。

〔プログラム 1〕

```
import java.util.Vector;

public class CallCenter {
   private final Vector waitingList = new Vector();
   private boolean running; // 呼出し生成中は true

   public static void main(String[] args) {
      int op = 2; // オペレータ数
      // 通話時間(秒)
      long[] duration = {130, 100, 150, 90, 110, 140};
      long interval = 60; // 呼出しを発生させる間隔(秒)
      new CallCenter(op, duration, interval);
   }

   public CallCenter(int op, long[] duration, long interval) {
      running = true;
      // オペレータのスレッドを生成し，開始する。
      for (int i = 0; i < op; i++) new Operator().start();
      long nextCallTime = System.currentTimeMillis();
      for (int i = 0; i < duration.length; i++) {
         // 呼出しを一つ生成し，リストに追加する。
         synchronized (waitingList) {
            waitingList.add(new Call(duration[i]));
            waitingList.notify();
         }
```

```
            // 次の呼出しを生成するまで待つ。
            nextCallTime += interval * 100;        // 10 倍の速さで動作
            long sleeping = [      a      ];
            try {
               if (sleeping > 0) Thread.sleep(sleeping);
            } catch (InterruptedException ie) {}
         }
         // すべてのオペレータのスレッドを終了させる。
         running = false;
         [      b      ] {
            waitingList.notifyAll();
         }
      }

      Call answer() {
         synchronized (waitingList) {
            while (waitingList.isEmpty() [      c      ]) {
               try {
                  [      d      ];
               } catch (InterruptedException ie) {}
            }
            if (waitingList.isEmpty()) return null;
            return (Call)waitingList.remove(0);
         }
      }

      class Operator extends Thread {
         public void run() {
            Call call;
            [      e      ] call.talk();
         }
      }
   }
```

〔プログラム 2〕

```
public class Call {
   private final long start, duration;
   public Call(long duration) {
      this.duration = duration;
      start = System.currentTimeMillis();
   }
   public void talk() {
      long elapsed = System.currentTimeMillis() - start;
      // 経過時間を四捨五入して表示する。
      System.out.println([      f      ] + "(s)");
      try {
         Thread.sleep(duration * 100);        // 10 倍の速さで動作
      } catch (InterruptedException ie) {}
   }
}
```

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　nextCallTime

イ　nextCallTime - duration[i] * 100

ウ　nextCallTime - System.currentTimeMillis()

エ　System.currentTimeMillis() - nextCallTime

b に関する解答群

ア　for (int i = 0; i < op; i++)

イ　if (waitingList != null)

ウ　synchronized (this)

エ　synchronized (waitingList)

c に関する解答群

ア　&& !running

イ　&& running

ウ　== !running

エ　|| running

d に関する解答群

ア　notify()

イ　wait()

ウ　waitingList.notify()

エ　waitingList.wait()

e に関する解答群

ア　if ((call = answer()) != null)

イ　if (answer() != null)

ウ　while ((call = answer()) != null)

エ　while (answer() != null)

fに関する解答群

ア (elapsed + 50) / 100　　イ (elapsed + 500) / 100

ウ (elapsed - 50) / 100　　エ (elapsed - 500) / 100

設問2 CallCenterのインスタンスを，次に示すようにして生成したとき，インスタンスが生成されてからシミュレータの時間で80秒後（実時間で8秒後）に通話中の状態にあるオペレータの人数を，解答群の中から選べ。

```
new CallCenter(3, new long[]{70, 90, 100, 110}, 30);
```

解答群

ア 0　　イ 1　　ウ 2　　エ 3

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1～4に答えよ。

〔プログラムの説明〕

副プログラム TIME は，時刻1と時刻2の差を求める。ここで，時刻は図1に示す形式の文字列とする。

h	h	m	m	s	s

図1　時刻の形式

hh は時，mm は分，ss は秒を表す。00 ≦ hh ≦ 24，00 ≦ mm ≦ 59，00 ≦ ss ≦ 59 とする。

(1) GR1 ～ GR3 には，それぞれ次の内容が設定されて，主プログラムから呼ばれる。

GR1 ：時刻1の先頭アドレス

GR2 ：時刻2の先頭アドレス

GR3 ：結果（時刻2 － 時刻1）を格納する領域の先頭アドレス

(2) 時刻1と時刻2は同一日付の時刻とし，000000 ≦ 時刻1 ＜ 時刻2 ≦ 240000 である。

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

〔プログラム〕

(行番号)

```
 1   TIME     START
 2            RPUSH
 3            LD       GR4,=5        ; ループカウンタ
 4            LD       GR7,=0        ; 下位けたへのけた下がりの調整値
 5            ADDL     GR1,GR4       ; GR1 ←時刻 1 の最終けたのアドレス
 6            ADDL     GR2,GR4       ; GR2 ←時刻 2 の最終けたのアドレス
 7            ADDL     GR3,GR4
 8   LOOP     LD       GR5,0,GR1     ; GR5 ←時刻 1 の 1 けた
 9            LD       GR6,0,GR2     ; GR6 ←時刻 2 の対応けた
10            SUBA     GR6,GR7       ; 下位けたへのけた下がりの調整
11            CPA      GR6,GR5
12            [    a    ]
13            LD       GR7,=0
14            JUMP     CALC
15   ADJUST   ADDA     GR6,ADD,GR4   ; 上位けたの 1 に相当する値を加算
```

```
16                  [        b        ]
17    CALC      SUBA    GR6,GR5
18              OR      GR6,='0'      ; 文字コードに変換
19              ST      GR6,0,GR3
20              SUBA    GR4,=1
21              JMI     FINISH
22              LAD     GR1,-1,GR1
23              LAD     GR2,-1,GR2
24              LAD     GR3,-1,GR3
25              JUMP    LOOP
26    FINISH    RPOP
27              RET
28    ADD       DC      0,10,6,10,6,10
29              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。

解答群

ア	ADDA	GR7,=1	イ	JMI	ADJUST
ウ	JNZ	ADJUST	エ	JPL	ADJUST
オ	JZE	ADJUST	カ	LD	GR7,=-1
キ	LD	GR7,=1	ク	SUBA	GR7,=1

設問2　時刻1と時刻2が次のとおりであるとき，行番号15のADDA命令は何回実行されるか。正しい答えを，解答群の中から選べ。

時刻1：　143726

時刻2：　170125

解答群

ア　1　　イ　2　　ウ　3　　エ　4　　オ　5

カ　6

設問3　次に示す形式の8文字からなる時刻を処理できるように，副プログラム TIME の行番号3と行番号28を変更した。［　　］に入れる正しい答えを，解答群の中から選べ。

h	h	m	m	s	s	d	d

ここで，dd は百分の一秒を表す。00 ≦ dd ≦ 99 とする。1日の終わりは 24000000 である。

〔プログラムの変更〕

(行番号)

```
                ⋮
 3              LD      GR4,=7          ; ループカウンタ
                ⋮
28    ADD       DC      0,10,6,10,6,10,［　　］
                ⋮
```

解答群

ア　6,6　　イ　6,10　　ウ　10,6　　エ　10,10

オ　10,100　　カ　100,10　　キ　100,100

アセンブラ

設問4　副プログラム TIME（設問3による変更の前）を使用して，駅伝競走のある区間の最高タイム（最小の所要時間）とそのチーム番号を出力する副プログラム EKIDEN を作成した。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。

(1) チーム数は9で，チーム番号は1～9である。

(2) 図2に示すとおりに設定された領域の先頭アドレスが GR1 に格納されて，主プログラムから呼ばれる。図2のスタート時刻とゴール時刻は，図1に示す形式とする。

	0 ～ 5	6 ～ 11
(GR1)＋ 0	チーム1のスタート時刻	チーム1のゴール時刻
＋12	チーム2のスタート時刻	チーム2のゴール時刻
⋮	⋮	⋮
＋96	チーム9のスタート時刻	チーム9のゴール時刻

図2　副プログラム EKIDEN の引数

(3) 出力形式は次のとおりとする。

n△hhmmss

n はチーム番号を表す文字，△は間隔文字とする。

(4) 同タイムのチームが複数ある場合は，同タイムのチームのうち最小のチーム番号を出力する。

(5) 全チームとも，スタートした日のうちにゴールするものとする。

(6) 副プログラム EKIDEN から戻るとき，GR1 ～ GR7 の内容は元に戻す。

```
EKIDEN  START
        RPUSH
        LD      GR7,=5          ;
INIT    LD      GR0,MAX,GR7     ;
        ST      GR0,BEST,GR7    ;  最高タイムの初期値を設定
        SUBA    GR7,=1          ;
        JMI     BEGIN           ;
        JUMP    INIT            ;
BEGIN   LD      GR7,='1'        ; チーム番号
```

```
LOOP1   LAD     GR4,BEST        ; 最高タイム
        LAD     GR3,CTIME       ; 処理中のチームのタイムを設定する領域
        LAD     GR2,6,GR1       ; 処理中のチームのゴール時刻
        CALL    TIME            ; 処理中のチームのタイムを計算
        LD      GR5,=6          ; ループカウンタ
LOOP2   LD      GR0,0,GR3
        CPL     GR0,0,GR4       ; 処理中のチームのタイムと最高タイムを比較
        [   c   ]
        JPL     NEXT            ; 処理中のチームのタイム > 最高タイム
        LAD     GR3,1,GR3       ; 処理中のチームの次のけた
        LAD     GR4,1,GR4       ; 最高タイムの次のけた
        SUBA    GR5,=1
        JNZ     LOOP2
        JUMP    NEXT            ; 処理中のチームのタイム = 最高タイム
REP     ST      GR0,0,GR4       ;
        LAD     GR3,1,GR3       ;
        LAD     GR4,1,GR4       ;  } 処理中のチームのタイムを
        LD      GR0,0,GR3       ;                最高タイムとする
        SUBA    GR5,=1          ;
        [   d   ]               ;
        ST      GR7,TEAM        ; 処理中のチームの番号を出力領域に設定
NEXT    CPA     GR7,='9'        ; 全チームの処理終了？
        JZE     FIN
        LAD     GR1,12,GR1      ; ポインタを次のチームへ進める
        ADDA    GR7,=1          ; チーム番号を表す文字コードに 1 を加算
        JUMP    LOOP1
FIN     OUT     TEAM,LENG
        RPOP
        RET
LENG    DC      8
TEAM    DS      1
        DC      ' '
BEST    DS      6
MAX     DC      '999999'
CTIME   DS      6
        END
```

解答群

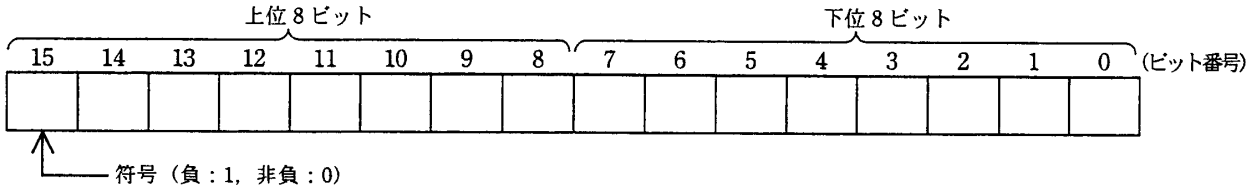
ア	JMI LOOP1	イ	JMI REP	ウ	JPL LOOP1
エ	JPL REP	オ	JZE LOOP1	カ	JZE REP

■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	(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
		r,adr [,x]		
論理比較 ComPare Logical	CPL	r1,r2		
		r,adr [,x]		

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

(4) シフト演算命令

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

(5) 分岐命令

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

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

(6) スタック操作命令

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

(7) コール，リターン命令

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

(8) その他

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

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

1.3 文字の符号表

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

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

2. アセンブラ言語 CASLⅡの仕様

2.1 言語の仕様

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

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

(注) [ ] [ ] 内の指定が省略できることを示す。
{ } { } 内の指定が必須であることを示す。

ラベル　その命令の（先頭の語の）アドレスを他の命令やプログラムから参照するための名前である。長さは 1～8 文字で，先頭の文字は英大文字でなければならない。以降の文字は，英大文字又は数字のいずれでもよい。なお，予約語である GR0～GR7 は，使用できない。

空白　1文字以上の間隔文字の列である。

命令コード　命令ごとに記述の形式が定義されている。

オペランド　命令ごとに記述の形式が定義されている。

コメント　覚え書きなどの任意の情報であり，処理系で許す任意の文字を書くことができる。

2.2 命令の種類

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

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

2.3 アセンブラ命令

アセンブラ命令は，アセンブラの制御などを行う。

(1)

START	[実行開始番地]

START 命令は，プログラムの先頭を定義する。

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

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

(2) | END | |

END 命令は，プログラムの終わりを定義する。

(3) | DS | 語数 |

DS 命令は，指定した語数の領域を確保する。

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

(4) | DC | 定数［,定数］… |

DC 命令は，定数で指定したデータを（連続する）語に格納する。

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

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

2.4 マクロ命令

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

(1) | IN | 入力領域,入力文字長領域 |

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

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

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

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

(2) | OUT | 出力領域,出力文字長領域 |

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

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

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

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

(3) | RPUSH |

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

(4) | RPOP |

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

2.5 機械語命令

機械語命令のオペランドは，次の形式で記述する。

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

2.6 その他

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

(2) 生成された命令語，領域は，主記憶上で連続した領域を占める。

3. プログラム実行の手引

3.1 OS

プログラムの実行に関して，次の取決めがある。

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

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

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

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

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

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

3.2 未定義事項

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

10. 答案用紙の記入に当たっては，次の指示に従ってください。

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

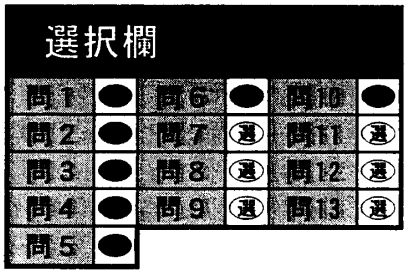
(2) 答案用紙は光学式読取り装置で処理しますので，答案用紙の**マークの記入方法**のとおりマークしてください。

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

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

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

〔**問6**と**問10**を選択した場合の例〕



(6) 解答は，次の例題にならって，解答欄にマークしてください。

〔例題〕　次の［　　］に入れる正しい答えを，解答群の中から選べ。

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

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。