平成２６年度

名古屋大学大学院情報科学研究科
情報システム学専攻
入学試験問題

専門

平成２５年８月７日（水）
１２：３０～１５：３０

注意事項

１．試験開始の合図があるまでは、この問題冊子を開いてはならない。

２．試験終了まで退出できない。

３．外国人留学生は英語で解答してよい。また、和英辞書などの辞書を１冊に限り使用してよい。電子辞書の持ち込みは認めない。

４．問題冊子、解答用紙３枚、草稿用紙３枚が配布されていることを確認せよ。

５．問題は(1)解析・線形代数、(2)確率・統計、(3)プログラミング、(4)計算機理論、(5)ハードウェア、(6)ソフトウェアの６科目がある。(4)~(6)の３科目から少なくとも１科目を選択して解答し、(1)~(3)を含めた６科目から合計３科目を選択して解答せよ。
なお、選択した科目名を解答用紙の指定欄に記入せよ。

６．解答用紙は指定欄に受験番号を必ず記入せよ。解答用紙に受験者の氏名を記入してはならない。

７．解答用紙に書ききれない場合は、裏面を使用してもよい。
ただし、裏面を使用した場合は、その旨、解答用紙表面右下に明記せよ。

８．解答用紙は試験終了後に３枚とも提出せよ。

９．問題冊子、草稿用紙は試験終了後に持ち帰ってよい。

解析・線形代数

（解の導出過程を書くこと）

[1] 以下の手順に従って，次の微分方程式の一般解を求める．

$$\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y = x^2 \tag{1}$$

(a) 微分方程式 $\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y = 0$ の一般解を求めよ．

(b) 微分方程式 (1) の特殊解を求めよ．

(c) 微分方程式 (1) の一般解を求めよ．

[2] 以下の手順に従って，$z^2 = 1 - i$ を満たすような複素数 z を求める．

(a) $z = |z|e^{i\theta}$ とおくとき，z^2 を極形式で表せ．
ここで，$|z|$ は z の絶対値，θ は z の偏角である．

(b) $1 - i$ を極形式で表せ．

(c) $|z|$ を求めよ．

(d) θ を $-\pi \leq \theta < \pi$ の範囲で，全て求めよ．

(e) 全ての z を極形式で表すとともに，複素平面上に図示せよ．

[3] 次の漸化式について考える．

$$x_{n+2} = 3x_{n+1} - 2x_n \ (n = 0, 1, 2, ...)$$

このとき，以下の問いに答えよ．

(a) $\boldsymbol{y}_n = \begin{pmatrix} x_n \\ x_{n+1} \end{pmatrix}$ とするとき，$\boldsymbol{y}_{n+1} = \boldsymbol{A}\boldsymbol{y}_n$ となるような行列 $\boldsymbol{A}$ を求めよ．

(b) 行列 $\boldsymbol{A}$ の固有値と，各固有値に対応する固有ベクトルを求めよ．

(c) 行列 $\boldsymbol{A}$ は，ある正則行列 $\boldsymbol{P}$ によって，$\boldsymbol{D} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}$ と対角化される．
このような行列 $\boldsymbol{D}, \boldsymbol{P}, \boldsymbol{P}^{-1}$ を求めよ．

(d) x_0, x_1 を用いて，x_n の一般形を示せ．

(e) $x_0 = 0, x_1 = 1$ とするとき，x_{10} の値を 4 桁の数字で答えよ．

Translations of technical terms

微分方程式	differential equation	行列	matrix
一般解	general solution	固有値	eigenvalue
特殊解	particular solution	固有ベクトル	eigenvector
複素数	complex number	正則行列	nonsingular matrix
極形式	polar form	対角化	diagonalization
絶対値	absolute value	一般形	general form
偏角	argument	値	value
複素平面	complex plane	桁	digit
漸化式	recurrence relation		

確率・統計　（解の導出過程も書くこと.）

［1］　以下の問いに答えよ.

(1) さいころ(dice)を2回投げるときに,2回目の目の数が1回目の目の数より大きい確率を求めよ.

(2) さいころを2回投げるときに，2回の目のうち最大の目の数がkである確率を求めよ.

(3) さいころを4回投げるときに，4回の目のうち最大の目の数がkである確率を求めよ.

［2］　確率変数X,Yの同時確率密度関数が$f_{X,Y}(x,y)=\dfrac{1}{2\pi}e^{-(x^2+y^2)/2}$　である. 以下の問いに答えよ.

（導出過程で，$\int_{-\infty}^{\infty}e^{-x^2}dx=\sqrt{\pi}$　を使っても良い.）

(1) 周辺確率密度関数$f_X(x)$を求めよ.

(2) 確率変数X,Yは独立であるかどうかを，その理由と共に述べよ.

(3) 確率変数$Z=X+Y$の確率密度関数$g_Z(z)$を求めよ.

［3］　ある機械で作られた部品の重さは，平均μ[kg]，標準偏差 0.5[kg]の正規分布に従うことがわかっている. 以下の問いに答えよ.

但し，標準正規分布を$f(x)$としたとき，$\int_{-2.58}^{2.58}f(x)dx=0.99$とする.

(1) 部品の中からn個の標本を無作為に取り出し，その重さの標本平均を求めた. この標本平均はどのような分布に従うか答えよ. また，標本平均の平均，標準偏差を答えよ.

(2) 25個の標本を取り出し,その重さの平均を求めたところ,4.0[kg]であった. μを99%の信頼水準で区間推定せよ.

【専門用語の英訳】

確率 probability, 確率変数 random variable, 同時確率密度関数 joint probability density function, 周辺確率密度関数 marginal probability density function, 独立 independent, 確率密度関数 probability density function, 平均 mean, 標準偏差 standard deviation, 正規分布 normal distribution, 標準正規分布 standard normal distribution, 標本 sample, 信頼水準 confidence level, 区間推定 interval estimation

プログラミング

与えられた正整数 (positive integer) N に対し，1 以上 N 以下の正整数を要素 (element) とし，それらが 0 個以上重複なくつながれた双方向リスト (doubly linked list) を考える．この双方向リストを，名前を list，要素数 (array size) を N+2 とする構造体の配列 (array of structures) により表現する．双方向リスト上の要素 i (1≦i≦N) を配列の要素 list[i] に対応させる．また，list[0] は双方向リストの先頭要素を指すために，list[N+1] は末尾要素を指すために使用される．これに対応して HEAD=0，TAIL=N+1 と定義する．構造体のメンバ (member) は next と prev とし，next は双方向リスト上の次の要素または TAIL，prev は双方向リスト上の前の要素または HEAD を保持し，以下の条件 1 を満足する．

条件 1：メンバ next を使うことにより，HEAD から順方向 (forward direction) に双方向リスト上のすべての要素をたどることができ，最終的に TAIL に到達する．また，メンバ prev を使うことにより，TAIL から逆方向 (backward direction) に双方向リスト上のすべての要素をたどることができ，最終的に HEAD に到達する．

表 1 は，N=8 として要素 1，5，3 がこの順でつながれた双方向リストを表現したときの構造体配列 list の各要素のメンバ next，prev の値である．要素 2，4，6，7，8 は未使用 (unused) であり，表中の「−」は任意の値である．また，プログラムリスト 1 は N=8 として双方向リストの操作を C 言語で記述したプログラムである．表 2 に main 関数を除く各関数(function)の機能 (functionality) を示す．

表 1. N=8 として要素 1，5，3 がこの順でつながれた双方向リストを表現したときの構造体配列 list の各要素のメンバ next, prev の値

要素	0(=HEAD)	1	2	3	4	5	6	7	8	9(=TAIL)
next	1	5	−	9	−	3	−	−	−	−
prev	−	0	−	5	−	1	−	−	−	3

表 2. プログラムリスト 1 における関数の機能

関数名	関数の機能
init_list()	初期化(initialization)
insert_elem(elem)	要素 elem を双方向リストの先頭要素となるように挿入 (insertion)
delete_elem(elem)	要素 elem を双方向リストから削除 (deletion)
print_list()	双方向リスト上の要素の出力 (output)

これに対して以下の問いに答えよ.

(1) プログラムリスト1の関数 delete_elem(elem)は, 双方向リストにおける要素 elem の次(要素または TAIL)と前(要素または HEAD)を接続することにより実現できる. プログラムリスト1の(a)の部分に C 言語の代入文 (assignment statement)を 5 文以内で追加し, プログラムを完成させよ. なお, ここでは引数 (argument) elem は双方向リスト上にある要素であることを仮定する.

(2) プログラムリスト1の 45 行目終了時の list[8].next と list[8].prev, 48 行目終了時の list[HEAD].next の値をそれぞれ書け.

(3) プログラムリスト1の 46 行目, 51 行目の print_list() 関数によって出力される文字列をそれぞれ書け.

プログラムリスト1の関数 main()内にある各関数の実行終了後において, 構造体配列 list は上記条件1を満足する. 以下の問いでは, 関数の実行終了後において上記条件1を満足しない場合を不具合 (*bug*) とよぶ.

(4) プログラムリスト1では, 双方向リスト上に存在する要素 e に対して関数 insert_elem(e) を実行すると不具合を生じる場合がある. このような不具合が生じる場合について, 具体的な例を用いて一つ説明せよ.

(5) プログラムリスト1では, 双方向リスト上に存在しない要素 f に対して関数 delete_elem(f) を実行すると不具合を生じる場合がある. このような不具合が生じる場合について, 具体的な例を用いて一つ説明せよ.

(6) 上記(4)(5)の不具合を防ぐため, 双方向リスト上に存在する要素 e に対する関数 insert_elem(e)の実行, ならびに双方向リスト上に存在しない要素 f に対する関数 delete_elem(f)の実行に対して, 双方向リストを変更することなく関数を終了するようにしたい. そのためのプログラムリスト1に対する変更方針を以下の4項目に分け, それぞれ簡潔に説明せよ. 変更する必要がない項目には「変更なし」と記載せよ.
(a) データ構造 (data structure)や格納値の意味 (meanings of stored values) の変更,
(b) 関数 insert_elem(elem)内の変更,
(c) 関数 delete_elem(elem)内の変更,
(d) その他の関数の変更.

(7) プログラムリスト1のメンバ prev を使用せずに実現した単方向リスト(singly linked list) と比較したときの双方向リストのメリットとデメリットをそれぞれ簡潔に説明せよ. ただし, 日本語で解答するならばそれぞれ 40 文字以内、英語で解答するならばそれぞれ 30 words 以内で記述せよ.

プログラムリスト１（行頭の数字は行番号を表す）

```
1: #include <stdio.h>

2: #define N 8
3: #define HEAD 0
4: #define TAIL ((N)+1)

5: typedef struct _list {
6:     int prev;
7:     int next;
8: } DLLIST;
9: DLLIST list[N+2];

10: void init_list(void) {
11:     list[HEAD].next = TAIL;
12:     list[TAIL].prev = HEAD;
13: }

14: void insert_elem(int elem) {
15:     int next;
16:     if((elem <= HEAD) || (elem >= TAIL)) return;
17:     next = list[HEAD].next;
18:     list[elem].next = next;
19:     list[next].prev = elem;
20:     list[HEAD].next = elem;
21:     list[elem].prev = HEAD;
22: }

23: void delete_elem(int elem) {
24:     int next, prev;
25:     if((elem <= HEAD) || (elem >= TAIL)) return;
        +-------------------+
        |        (a)        |
        +-------------------+
26: }

27: void print_list(void) {
28:     int i;
29:     i = list[HEAD].next;
30:     if(i == TAIL) {
31:             printf("EMPTY¥n");
32:     } else {
33:             printf("%d", i);
                                                    /* 次頁に続く */
34:             for(i=list[i].next; i!=TAIL; i=list[i].next) {
```

```
35:                     printf("->%d", i);
36:             }
37:             printf("¥n");
38:     }
39: }

40: int main(void) {
41:     init_list();

42:     insert_elem(3); print_list();
43:     insert_elem(5); print_list();
44:     insert_elem(8); print_list();
45:     delete_elem(5); print_list();
46:     insert_elem(7); print_list();
47:     delete_elem(3); print_list();
48:     delete_elem(7); print_list();
49:     insert_elem(1); print_list();
50:     delete_elem(1); print_list();
51:     delete_elem(8); print_list();

52:     return 0;
53: }
```

計算機理論

[1] 記号列(string) w に含まれる記号(symbol) の出現回数(number of occurrences) を $||w||$ と表し，w の長さ (length) とよぶ．また w に現れる記号 a の出現回数を $||w||_a$ と表す．例えば，$||aaba|| = 4$, $||aaba||_a = 3$, $||aaba||_b = 1$ である．以下では，アルファベット (alphabet) を $\{a, b\}$ とする．

(1) 次の言語(language) L_1 を受理(accept) する決定性有限オートマトン (deterministic finite automaton, 以下 DFA) の状態遷移図(transition diagram) を書け．

$$L_1 = \{\ w \mid\ ||w||\ \text{は}\ 3\ \text{の倍数(multiple) でない}\ \}$$

(2) 次の言語 L_2 を受理する DFA の状態遷移図を書け．

$$L_2 = \{\ w \mid\ (3||w||_a - ||w||_b)\ \text{は}\ 5\ \text{の倍数である}\ \}$$

(3) $L_1 \cap L_2 = \emptyset$ は成り立つか．成り立つならばそれを証明し，そうでなければ反例 (counterexample) を示せ．ただし $\emptyset$ は空集合(empty set) を表す．

(4) k を自然数(natural number) とする．DFA M が与えられたとき，「M によって受理されるすべての記号列の長さが k の倍数かどうか」を判定(decide) するアルゴリズム (algorithm) を書け．ただし，次の事実 1，事実 2 を使ってよい．

事実 1 L, L' がそれぞれ DFA M, M' によって受理されるとき，$L \cap L'$ を受理する DFA を M, M' から構成するアルゴリズム A_1 が存在する．

事実 2 DFA M が与えられたとき，M が受理する言語が空集合かどうかを判定するアルゴリズム A_2 が存在する．

[2] 以下の一階述語論理式(first-order predicate logic formula) がそれぞれ恒真(valid) であるか否かを述べよ．また，その理由を示せ．

ただし，P, Q, R は述語記号(predicate symbol) とする．また，論理記号(logical symbol) $\rightarrow$ は右結合(right associative) とし，論理記号の優先順位(precedence) は高い順から $\forall$, $\exists$, $\wedge$, $\vee$, $\rightarrow$ とする．

(1) $\forall x.\forall y.\forall z.(P(x) \wedge P(y) \wedge P(z) \rightarrow R(x,y) \vee R(x,z) \vee R(y,z))$
$\qquad \rightarrow \forall x.\forall y.(P(x) \wedge P(y) \rightarrow R(x,y))$

(2) $\forall x.\forall y.(P(x) \wedge P(y) \rightarrow R(x,y))$
$\qquad \rightarrow \forall x.\forall y.\forall z.(P(x) \wedge P(y) \wedge P(z) \rightarrow R(x,y) \vee R(x,z) \vee R(y,z))$

(3) $\exists x.((\forall x.(P(x) \rightarrow Q(x)) \rightarrow \exists x.P(x)) \rightarrow P(x))$

(4) $\exists x.(\forall x.(P(x) \rightarrow Q(x) \rightarrow P(x)) \rightarrow P(x))$

ハードウェア

[1] 回路面積（circuit area）が30のプロセッサAで，プログラムXを実行した時にかかる実行時間（execution time）を100，消費エネルギー（energy consumption）を100とする。プロセッサAでプログラムXを実行した時に，各命令クラス（instruction class）に属する命令の実行時間および消費エネルギーが，プログラムX全体の実行時間および消費エネルギーに占める割合は表1の通りである。

このプロセッサAを改善する手法として，表2の5つの手法が適用可能であることがわかっている。ただし，手法1と手法2は同時に適用することができない。表2には，それぞれの手法による実行時間の増減，消費エネルギーの増減，回路面積の増減も示している。

ここで，プロセッサAは命令（instruction）を逐次的に（sequentially）実行するため，各命令の実行時間および消費エネルギーが増減した分，プログラムX全体の実行時間および消費エネルギーも増減するものとする。また，システム全体が単一クロック（single clock）で動作しており，各改善手法の消費エネルギーの増減および回路面積の増減は，互いに独立であるとする。さらに，プロセッサAは，ロードストアアーキテクチャ（load store architecture）を採用しているものとする。

表1. 各命令クラスが実行時間と消費エネルギーに占める割合

命令クラス	実行時間に占める割合	消費エネルギーに占める割合
加減算（add and subtract）命令	25%	10%
乗除算（multiply and divide）命令	15%	25%
分岐（branch）命令	10%	10%
ロード（load）命令	15%	25%
ストア（store）命令	10%	20%
その他の命令	25%	10%

表2. プロセッサAに適用可能な改善手法

手法	実行時間の増減	消費エネルギーの増減	回路面積の増減
手法1	乗除算命令の実行時間が1/3になる	乗除算命令の消費エネルギーが2倍になる	5増加
手法2	乗除算命令の実行時間が2倍になる	乗除算命令の消費エネルギーが1/2になる	3減少
手法3	分岐命令の実行時間が1/2になる	分岐命令の消費エネルギーが2倍になる	5増加
手法4	ロード命令とストア命令の実行時間が2/3になる	ロード命令とストア命令の消費エネルギーが2倍になる	6増加
手法5	クロック周波数が1.1倍になる	システム全体の消費エネルギーが1.1倍になる	7増加

これに関して，以下の問いに答えよ。

(1) 回路面積を 40 以下に抑えるという条件下で，プログラム X の実行時間を最短にするようプロセッサ A を性能向上させるためには，どの手法（1 つまたは複数）を採用すればよいか。また，その時のプログラム X の実行時間と消費エネルギーはいくらになるか。

(2) プログラム X 全体の消費エネルギーを 150 以下に抑えるという条件下で，プログラム X の実行時間を最短にするようプロセッサ A を性能向上させるためには，どの手法（1 つまたは複数）を採用すればよいか。また，その時のプログラム X の消費エネルギーと回路面積はいくらになるか。

(3) プロセッサ A にデータキャッシュ（data cache）を追加すると，どの命令クラスの実行時間がどのように変化するか。また，消費エネルギーがどのように変化するかについて考察せよ。なお，データキャッシュは，ライトバック（write back）方式を採用しているものとする。

[2] 単一クロック（single clock）のプロセッサ（processor）に対して，図 1 に示すデータパス（data path）と図 2 に示す 2 つの命令タイプ（instruction type）が与えられたとき，以下の問いに答えよ。

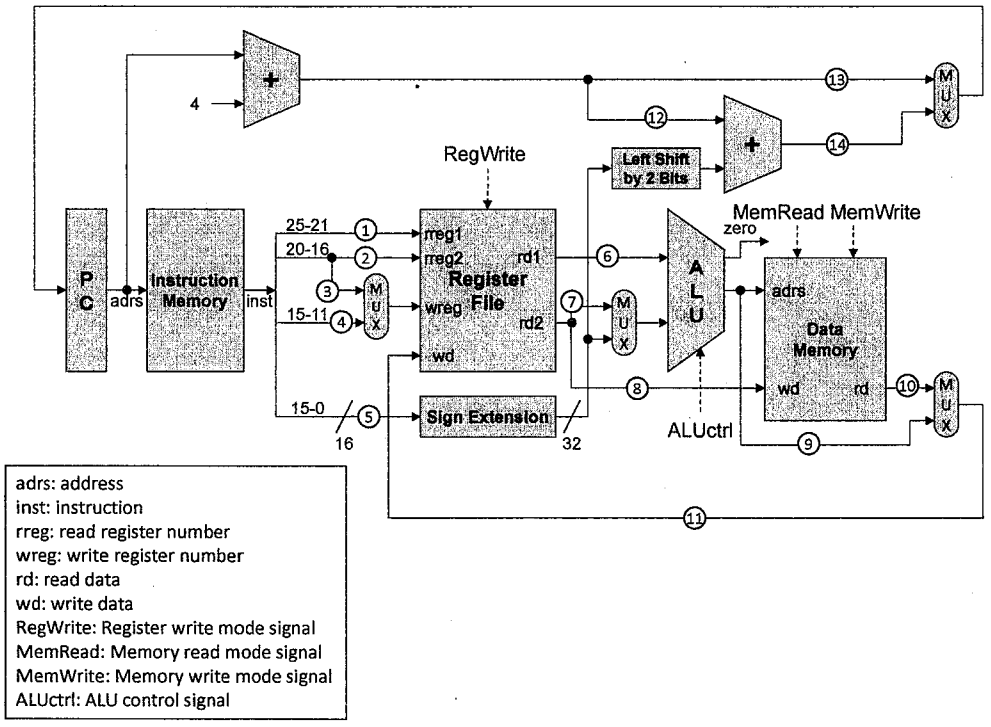

図 1. データパス

(1) このプロセッサの加算命令（add instruction）は R-Type であり，rs の値と rt の値を足した値を rd に格納する。演算において，shamt と funct は使用しないものとする。この命令が実行された場合に，有効となる（effective）データパスに含まれるものを①～⑭からすべて選べ。解答欄には数字を小さい順に並べること。

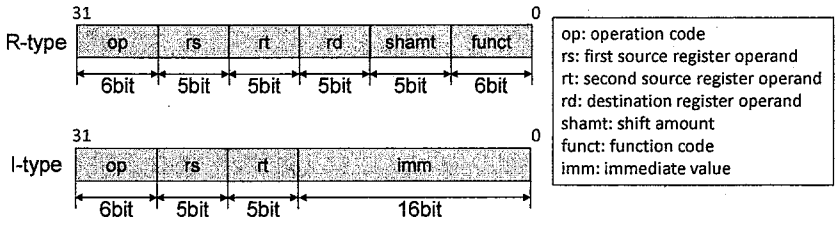

図 2. 命令タイプ

（2）このプロセッサのストアワード命令（store word instruction）はI-Typeであり，rsの値をベース（base），immをオフセット（offset）として得られる番地（address）に，rtの値を書き込む。この命令が実行された場合に，有効となるデータパスに含まれるものを①～⑭からすべて選べ。解答欄には数字を小さい順に並べること。

（3）このプロセッサの分岐命令（branch instruction）はI-Typeであり，rsの値とrtの値が等しい場合に分岐条件（branch condition）が成立し，分岐命令の次の命令の番地にimmを4倍した値を足した番地にジャンプ（jump）する。この命令が実行され，分岐条件が成立した場合に，有効となるデータパスに含まれるものを①～⑭からすべて選べ。解答欄には数字を小さい順に並べること。

ソフトウェア

[1] 並行プロセス (concurrent process) として生産者/消費者問題(producer/consumer problem) を考える。図 1 に示す producer と consumer で定義されるプロセスが 1 つずつ単一プロセッサ (uni-processor) 上で並行に実行されているとする。producer が getdata() によって取得したデータを buffer に書きこみ，consumer が buffer のデータを読みだして print() によって出力することを意図している。ここで，データを取得する関数 getdata() と，データを出力する関数 print() はそれぞれ他で定義されているものとする。変数 count は共有変数(shared variable) であり，0 に初期化(initialize) されている。sleep() を実行したプロセスは，待機状態(waiting state) となり，wakeup(consumer)，wakeup(producer) はそれぞれ引数に持つプロセスが待機状態にあるとき，待機状態から実行可能状態(ready state) に状態を変化させる。

```
 1:  #define N 100
 2:  #define TRUE 1
 3:
 4:  int count = 0;
 5:  int buffer[N];
 6:
 7:  void producer(void) {
 8:    int p1; p1=0;
 9:    while(TRUE) {
10:      if (count == N) sleep();
11:      buffer[p1] = getdata(); p1=p1+1;
12:      if(p1>=N) p1=0;
13:      count = count + 1;
14:      if (count == 1) wakeup(consumer);
15:    }
16:  }
17:  void consumer(void) {
18:    int p2; p2=0;
19:    while(TRUE) {
20:      if (count == 0) sleep();
21:      count = count - 1;
22:      print(buffer[p2]); p2 = p2 + 1;
23:      if(p2>=N) p2=0;
24:      if (count == N - 1) wakeup(producer);
25:    }
26:  }
```

図 1. 生産者/消費者問題 (行頭の数字は行番号を示す。)

このとき以下の問いに答えよ。

(1) 図 1 において，共有変数 count に対して競合(conflict) が発生する。変数に対する競合とは何か簡単に説明せよ。

(2) プロセス producer が実行されず，プロセス consumer が，一回目に実行されたとき，20 行目において，count の値を読みこんだ直後にスケジューラ (scheduler) によって実行状態(running state) から実行可能状態となり，sleep() を実行する前に producer に実行が移ると，公平なスケジューリング (fair scheduling) を行っても getdata による取得データが出力されることなく，両方のプロセスとも待機状態となってしまうことがある。この状況に至る理由を説明せよ。

(3) 上記のような状況を防ぐ手法としてセマフォ (semaphore) を用いた実行制御が行われる。計数セマフォ (counting semaphore) に対する P命令(wait instruction) と V命令(signal instruction) の仕組みについて説明せよ。

(4) 二値セマフォ (binary semaphore) mutex および計数セマフォ full，empty を用いて生産者/ 消費者問題を実現する。どのようなスケジューリングに対しても getdata() で取得したデータがすべて print() で出力されるように図 2 の (a)，(b)，(c)，(d)，(e)，(f)，(g)，(h) に mutex，full，empty に対する適当な P，V 命令を挿入せよ。ここで，セマフォ S に対する P 命令，V 命令はそれぞれ P(S)，V(S) と書くことにする。また，型 semaphore は整数型として定義されているとする。

```
 1:  #define N 100
 2:  #define TRUE 1
 3:
 4:  int count = 0;
 5:  int buffer[N];
 6:
 7:  semaphore mutex = 1;
 8:  semaphore empty = N;
 9:  semaphore full = 0;
10:
11:  void producer(void) {
12:    int p1; p1=0;
13:    while(TRUE) {
14:      [  (a)  ];
15:      [  (b)  ];
16:      buffer[p1] = getdata(); p1=p1+1;
17:      if(p1>=N) p1=0;
18:      count = count + 1;
19:      [  (c)  ];
20:      [  (d)  ];
21:    }
22:  }
```

```
23:  void consumer(void) {
24:    int p2; p2=0;
25:    while(TRUE) {
26:      [  (e)  ];
27:      [  (f)  ];
28:      count = count - 1;
29:      print(buffer[p2]); p2=p2+1;
30:      if(p2>=N) p2=0;
31:      [  (g)  ];
32:      [  (h)  ];
33:    }
34:  }
```

図 2 セマフォを用いた生産者/消費者問題 (行頭の数字は行番号を示す。)

[2] 次に示す文脈自由文法(context-free grammar) G について以下の問いに答えよ。

$$G = \langle N,\ T,\ P,\ S \rangle$$

ここで,

非終端記号集合(non-terminals) N $=$ $\{S\}$

終端記号集合(terminals) T $=$ $\{\mathrm{a},\ +,\ (,),\ \bullet,*\}$

生成規則集合(production rules) P $=$ $\left\{ \begin{array}{lll} S \to S{+}S, & S \to S \bullet S, & S \to (S), \\ S \to S*, & S \to \mathrm{a} & \end{array} \right\}$

開始記号(start symbol) S $\in$ N

とする。

(1) $(\mathrm{a}+\mathrm{a})*\bullet\mathrm{a}$ に対する最左導出(left-most derivation) を示し，構文解析木(parse tree) を示せ。

(2) G が曖昧である (ambiguous) ことを示せ。

(3) G に対して新たな非終端記号 A を一つだけ導入して，左再帰性(left-recursion) を除去した生成規則集合 P' を示せ。

上記で導入した A を非終端記号集合に追加し，生成規則集合を P' に修正した文法を

$$G_2 = \langle \{S,\ A\},\ T,\ P',\ S \rangle$$

とする。

(4) G_2 において，First(S)，First(A)，Follow(S)，Follow(A) を求めよ。ここで，入力の終了を示す記号は \$ を用いること。

(5) G_2 が $LL(1)$ 文法であるか否かを理由を示して答えよ。