システムプログラム・テスト問題用紙

('04 年 2 月 10 日（火）・ 8:50 〜 10:20)

解答上、その他の注意事項

I. 問題は、問 I〜V までである。

II. 解答用紙の右上の欄に学籍番号・名前を記入すること。

III. 解答欄を間違えないよう注意すること。

IV. 選択式でない問で解答欄がマス目になっている場合は、1 字に 1 マスを用いること。特に空白にも必ず 1 マスを用いること

V. 解答中の文字 (特に a と d) がはっきりと区別できるよう注意すること。

VI. 教科書・ノート・プリント・参考書などは持ち込み可である。
（ただし、プリントは机の上いっぱいに広げないこと。）

VII. パソコン・携帯電話などの通信機能を持つものは持ち込み不可 である。

VIII. テストの配点は 80 点である。（第 1 回レポート 10 点・第 2 回レポート 10 点）合格はレポートの得点を加点して、100 点満点中 60 点以上とする。

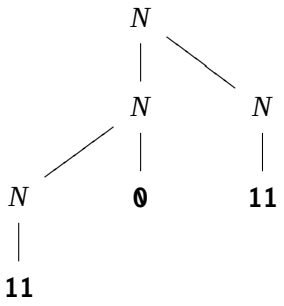
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$
\begin{array}{rcl}
N & \rightarrow & N\,\texttt{0} \\
 & | & N\,N \\
 & | & \texttt{11} \\
 & | & \texttt{1001}
\end{array}
$$

次の各文について、上の BNF の非終端記号 N から導出されるものには、その解析木 (parse tree) を右の例にならって書き、導出されないものには × を記せ。

例: 11011 に対する導出木



(1) 1110010　(2) 1001110　(3) 1100100

II. 「(b|abba)*」という正規表現に（一部でなく）全体がマッチする文字列には (A)、「(ba*b)*b」という正規表現に全体がマッチする文字列には (B)、両方に全体がマッチする文字列には (C)、どちらにもマッチしない文字列には (D) をつけよ。

(1) babbabb　(2) babbaab　(3) babaaba　(4) babbbab

III. （演算子順位法）

次の文法は、正規表現の一つの表記法を表している。

E → **id**
| E “++” E
| E “||” E
| E “*”
| “(” E “)”

ただし、この文法は曖昧なので、優先順位と結合性について次のように決めておく。

> 「++」と「||」は、いずれも左結合で、「*」は「++」 よりも優先順位が高く、「++」は「||」よりも優先順位が高いものとする。つまり、「a++b*」は 「a++(b*)」と解釈され、「a || b ++ c」は「a || (b ++ c)」と解釈される。

以下の演算子順位行列の空欄を ⋖, ≐, ⋗, **err** のいずれかで埋めよ。（**err** はエラーを表す。）

左 \ 右	||	++	*	(	)	**id**	終
始	⋖	⋖	⋖	⋖	**err**	⋖	≐
||	(1)	(2)	⋖	⋖	⋗	⋖	⋗
++	⋗	⋗	(3)	⋖	⋗	⋖	⋗
*	⋗	⋗	⋗	**err**	⋗	**err**	⋗
(	⋖	⋖	⋖	(4)	≐	⋖	**err**
)	⋗	⋗	⋗	**err**	⋗	**err**	⋗
id	⋗	⋗	⋗	**err**	⋗	**err**	⋗

IV. （再帰下降構文解析）

右のような BNF で表される文法は曖昧である。(··· の後のローマ数字は生成規則の番号)「**if**」,「**else**」,「**x**」,「**a**」はここでは終端記号と考える。「S」,「S'」は非終端記号で、開始記号は S である。
この文法に対して再帰的下向き構文解析ルーチンを作成する。

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{if x } S\ S' & \cdots\ I \\
 & | & \textbf{a} & \cdots\ II \\
S' & \rightarrow & \textbf{else } S & \cdots\ III \\
 & | & \epsilon & \cdots\ IV
\end{array}
$$

(1) **if x if x a else a** に対する解析木（parse tree）を 2 通り書け。

(2) *First*(S') を求めよ。

(3) *Follow*(S') を求めよ。

(4) この文法は曖昧なので、構文解析表を作成すると、あるエントリに 2 つの生成規則が入ってしまう。(つまり LL(1) ではない。) そのエントリとはどこか？下の選択肢から選べ。
(A) S の行と**if** の列の交点　　(B) S の行と$の列の交点
(C) S' の行と**else** の列の交点　　(D) S' の行と$の列の交点

(5) しかし、どちらかの生成規則を無理矢理選んで再帰的下向き構文解析ルーチンを作成することができる。下に示すのはそのプログラムである。この構文解析ルーチンは (1) の解答のどちらの解析木を生成することになるか。左または右で答えよ。

(6) プログラムの空欄(?) を埋めよ。

再帰下降構文解析プログラム

```
void S() {
  if (token=='a')  {
    eat('a');
  } else if (token==IF) {
      (?)
  } else {
    printf("構文に誤りがあります。¥n");
    exit (0); /* プログラムを終了 */
  }
}

void S1() {  /*  S' に対応する関数 */
  if (token==ELSE) {
    eat(ELSE); S();
  } else if (token==EOF) { /* 何もしない */
  } else {
    printf("構文に誤りがあります。¥n");
    exit (0); /* プログラムを終了 */
  }
}
```

(**プログラムの補足説明:** プログラム中では、終端記号は、「**a**」,「**x**」のような一文字の終端記号は、その字そのもの（の ASCII コード)、「**if**」,「**else**」は C 言語のマクロ IF, ELSE として表現している。入力の終り（**$**）はEOF というマクロで表す。

yylex 関数は、入力を読んで、次の終端記号を返す関数である。token という大域変数に、現在処理中の終端記号が代入される。eat 関数は、現在 token に入っている値が、引数として与えられた終端記号と等しいかどうか確かめ、等しければ次の終端記号を読み込む。)

（参考） 上のプログラムで省略された最初の部分

（問題には直接は関係ない。改ページの都合上いくつかに分割して示す。）

```
#include <stdio.h>
#include <stdlib.h>   /* exit() 用 */
#include <ctype.h>    /* isalpha() 用 */

#define IF    259
#define ELSE  261

/* 大域変数の宣言 */
int token;
int yylval;
```

```
int keyword(int c) {
  char buf[256];
  char* str;
  for(str = buf; isalpha(c) ;str++) {
      *str = c;
      c = getchar();
  }
  ungetc(c, stdin); /* c は入力に戻す */
  *str = '¥0'; /* 文字列の終を示す。*/

  if (strcmp(buf, "if")==0) return IF;
  else if (strcmp(buf, "else")==0) return ELSE;
  else if (strcmp(buf, "a")==0) return 'a';
  else if (strcmp(buf, "x")==0) return 'x';
}
```

```
int yylex() {   /* 簡易字句解析ルーチン */
  int c;

  do { /* 空白は読みとばす。*/
    c = getchar();
  } while (c == ' ' || c == '¥t' || c == '¥n');

  if (isalpha(c)) { /* アルファベットだったら... */
    keyword(c);
  } else if (c == EOF) { /* ファイルの終 */
    return EOF;
  } else { /* 上のどの条件にも合わなければ、文字をそのまま返す。*/
    return c;   /* '(', ')', ',' など */
  }
}
```

```
void eat(int t) {     /* token（終端記号）を消費して、次の token を読む */
  if (token == t) {
    token = yylex();
    return;
  } else {
    printf("構文に誤りがあります。¥n");
    exit (0);
  }
}
```

V. （LR 構文解析）

次のような正規表現を表す文法（··· の後は生成規則の番号）

$$
\begin{array}{lcll}
E & \rightarrow & \mathbf{a} & \cdots\ I \\
 & | & E\ \text{“|”}\ E & \cdots\ II \\
 & | & E\ E & \cdots\ III \\
 & | & E\ \text{“*”} & \cdots\ IV \\
 & | & \text{“(”}\ E\ \text{“)”} & \cdots\ V
\end{array}
$$

に対して、演算子の優先度、結合性を通常の正規表現と同じになるように指定して、`bison` に入力すると、`bison` の出力する LR 構文解析表は次のようになる。（**注:** `bison` に`-v` オプションを指定することによって、LR 構文解析表をファイルに出力させることができる。）
ただし、**a** は終端記号で、アルファベット 1 文字からなるトークンを表す。

	$	(	)	*	|	**a**	*E*
⓪		shift ②				shift ①	goto ③
①	reduce I						
②		shift ②				shift ①	goto ④
③	accept	shift ②		shift ⑥	shift ⑤	shift ①	goto ⑦
④		shift ②	shift ⑧	shift ⑥	shift ⑤	shift ①	goto ⑦
⑤		shift ②				shift ①	goto ⑨
⑥	reduce IV						
⑦	reduce III			shift ⑥	reduce III		goto ⑦
⑧	reduce V						
⑨	reduce II	shift ②	reduce II	shift ⑥	reduce II	shift ①	goto ⑦

ここで、shift ⓢは、「シフトして状態ⓢへ遷移」、 goto ⓢは、「状態ⓢへ遷移」、 reduce X は、「生成規則 X を使って還元」を表す。

次の入力に対して、↑の次（右）の記号を**シフトする直前**のスタックの状態（つまり直後の動作が還元ではなくシフトである状態）はどのようになっているか?

(1) `a*b*|cd*`　(2) `a*b*|cd*`　(3) `a*b*|cd*`　(4) `a*b*|cd*`

下の選択肢（(1)〜(4) 共通）から選べ。（左がスタックの底とする）

(A). ⓪ **a** ①　(B). ⓪ *E* ③　(C). ⓪ *E* ③ * ⑥
(D). ⓪ *E* ③ **a** ①　(E). ⓪ *E* ③ *E* ⑦　(F). ⓪ *E* ③ *E* ⑦ * ⑥
(G). ⓪ *E* ③ *E* ⑦ | ⑨ **a** ①　(H). ⓪ *E* ③ *E* ⑦ | ⑤ *E* ⑨　(I). ⓪ *E* ③ | ⑤
(J). ⓪ *E* ③ | ⑤ **a** ①　(K). ⓪ *E* ③ | ⑤ *E* ⑨　(L). （該当なし）

システムプログラム・テスト解答用紙('04年2月10日)

学籍番号		氏名	

I. (4×3)

(1).	(2).	(3).

II. (4×4)

(1).		(2).		(3).		(4).	

III. (3×4)

(1).		(2).		(3).		(4).	

IV. (4×6)

(1) その 1	(1) その 2
(2) { }	(3) { }
(4)	(5)
(6).	

V. (4×4)

(1)	(2)	(3)	(4)

テストの感想