コンパイラ（2017年度）・期末テスト問題用紙

（2017年08月03日（木）・ 10:30 〜 12:00）

解答上、その他の注意事項

I. 問題は、問I〜VIまである。

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. 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。

V. 持ち込みは不可である。筆記用具・時計・学生証以外のものは、かばんの中などにしまうこと。

VI. 期末テストの配点は80点である。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。

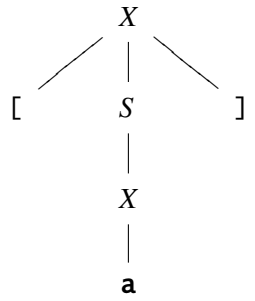
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$
\begin{array}{rcl}
X & \rightarrow & \text{“[”}\ S\ \text{“]”} \\
 & | & \mathbf{a} \\
S & \rightarrow & S\ \text{“;”}\ X \\
 & | & X
\end{array}
$$

ただし、X, S は非終端記号、“**[**”, “**a**”, “**]**”, “**;**” は終端記号である。次の各記号列について、上の BNF の非終端記号 X 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。（解析木は一通りとは限らないが、そのうち一つを書けば良い。）

例: `[a]` に対する解析木



(1) `[a;a]`　(2) `[[a]]`　(3) `[a;[a]]`　(4) `[[a];a]`

II. （正規表現）

以下の文字列について、「`x(z|yz|yy|yx)*x`」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 (L) を、「`xy(x|(y*z)|z)*y*yx`」 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 (R)、両方に全体がマッチする文字列には (B)、どちらにも全体がマッチしない文字列には (N) を記せ。

(1) **xyyzzyyx**　(2) **xyzzxyyx**　(3) **xyxyxyyx**　(4) **xyxyxxyx**

III. （コンパイラのフェーズ）

コンパイラは、字句（単語）を切り分ける字句解析フェーズ、プログラムの構造を木の形に表す構文解析フェーズ、変数の宣言や型のチェックを行なう意味解析（静的解析）フェーズ、目的のコードを生成するコード生成フェーズなどに概念的に分けることができる。

次の (1)〜(4) の C 言語のプログラムにはそれぞれ誤りがある。コンパイラのどのフェーズで誤りが検出されるか？（あるいはされないか？） もっとも適当なものを下の選択肢 (A)〜(E) から選べ。なお、(1)〜(4) のいずれも単独でコンパイルされ、標準ライブラリとのみリンクされるものとする。（つまり、他のファイルに変数や関数が定義されていることはない。）

(1) （波括弧と丸括弧を逆に使った）

```
#include <stdio.h>

int main{void} (
    printf{"Hello World!\n"};
    return 0;
)
```

(2) （printf 関数を引数なしで使用した。）

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!");
    printf();
    return 0;
}
```

(3) （文字列リテラルの途中で改行した。）

```
#include <stdio.h>

int main(void) {
    printf("Hello!
            World\n");
    return 0;
}
```

(4) （for 文のセミコロン; がひとつしかない。）

```
#include <stdio.h>

int main(void) {
    int i = 0;
    for (i < 5; i++) {
        printf("Hello World!\n");
    }
    return 0;
}
```

(1)〜(4) の選択肢

(A) 字句解析フェーズでエラーが検出される。

(B) 構文解析フェーズでエラーが検出される。

(C) 意味解析フェーズでエラーが検出される。

(D) コード生成フェーズでエラーが検出される。

(E) 実行時にエラーとなるか、全くエラーにならない（が作成者の意図と異なる動作をする）。

IV. （演算子順位法）

次の BNF で表される文法を演算子順位法により構文解析する。

$$E \rightarrow \quad \textbf{id} \;\mid\; E \text{ “**” } E \;\mid\; E \text{ “&” } E \;\mid\; E \text{ “\$\$” } E \;\mid\; E \text{ “<” } E \;\mid\; \text{“(” } E \text{ “)”}$$

ただし、**id** はアルファベット 1 文字からなるトークンを表す。

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。

「**」は右結合、「&」は左結合、「$$」は右結合、「<」は非結合、であり、「**」は「&」よりも優先順位が高く、「&」 は「$$」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、下表中の左の欄の式は、右の欄の式として解釈される。

式	解釈
a ** b ** c	a ** (b ** c)
a & b & c	(a & b) & c
a $$ b $$ c	a $$ (b $$ c)
a < b < c	（構文エラー）
a ** b & c	(a ** b) & c
a & b ** c	a & (b ** c)
a ** b $$ c	(a ** b) $$ c
a $$ b ** c	a $$ (b ** c)

式	解釈
a ** b < c	(a ** b) < c
a < b ** c	a < (b ** c)
a & b $$ c	(a & b) $$ c
a $$ b & c	a $$ (b & c)
a & b < c	(a & b) < c
a < b & c	a < (b & c)
a $$ b < c	(a $$ b) < c
a < b $$ c	a < (b $$ c)

以下の演算子順位行列の空欄 (1)〜(5) を ⋖、≐、⋗、✗ のうちもっとも適切なもので埋めよ。ただし ✗ はエラーを表すものとする。（教科書などの記法では、エラーは空欄のままとしているが、このテストでは無回答と区別するために明示的に ✗ を書くことにする。）

左 \ 右	**	&	$$	<	(	)	**id**	終
始	⋖	⋖	⋖	⋖	⋖	✗	⋖	≐
**	(1)	⋗	⋗	⋗	⋖	⋗	⋖	⋗
&	⋖	(2)	⋗	(3)	⋖	⋗	⋖	⋗
$$	⋖	⋖	(4)	⋗	⋖	⋗	⋖	⋗
<	⋖	⋖	⋖	(5)	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≐	⋖	✗
)	⋗	⋗	⋗	⋗	✗	⋗	✗	⋗
id	⋗	⋗	⋗	⋗	✗	⋗	✗	⋗

V. （再帰下降構文解析）

次のような BNF で定義された文法に対して再帰下降構文解析ルーチンを作成する。

$$
\begin{array}{lcll}
S & \to & \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S & \cdots\ \text{I} \\
 & | & \textbf{begin}\ L\ \textbf{end} & \cdots\ \text{II} \\
 & | & \textbf{write}\ E & \cdots\ \text{III} \\
F & \to & \textbf{id} & \cdots\ \text{IV} \\
 & | & \text{“(”}\ E\ \text{“)”} & \cdots\ \text{V} \\
L & \to & S & \\
 & | & L\ \text{“;”}\ S & \\
E & \to & F & \\
 & | & E\ \text{“+”}\ F & \\
 & | & E\ \text{“-”}\ F &
\end{array}
$$

ただし、「S」,「F」,「L」,「E」は非終端記号で、「**if**」,「**then**」,「**else**」,「**begin**」,「**end**」,「**write**」,「**id**」,「**(**」,「**)**」,「**;**」,「**+**」,「**-**」は終端記号とする。開始記号（start symbol）は S である。$\cdots$ の後のローマ数字（I, II など）は生成規則の番号である。

(1) L から左再帰を除去すると、次のような BNF が得られる。

$$
\begin{array}{lcll}
L & \to & S\ L' & \cdots\ \text{VI} \\
L' & \to & \varepsilon & \cdots\ \text{VII} \\
 & | & \text{“;”}\ S\ L' & \cdots\ \text{VIII}
\end{array}
$$

これを参考にして、E から左再帰を除去せよ。補助的に導入する非終端記号は E' とせよ。また、あとの問題の解答で使用するために、各生成規則に番号 (IX, X, XI, XII, ...) をつけておいてもよい。

以下の (2)〜(4) は、(1) で E と L から左再帰を除去して得られた BNF について答えよ。ただし、入力の終わりは $ で表せ。

(2) *Follow*(L') を求めよ。

(3) *Follow*(E') を求めよ。

(4) 下の予測型構文解析表の L, L' の行を埋めよ。この問題の解答は ✗, VI, VII, VIII の中から選べ。空欄のままにしないこと。ただし ✗ は “エラー” を示す。（教科書などの記法では、エラーは空欄のままとしているが、このテストでは無回答と区別するために明示的に ✗ を書くことにする。）

(5) 下の予測型構文解析表の S, F, E, E' の行を埋めよ。
ただし、この解答は、✗ と上記の BNF 中の生成規則の番号（I 〜V）と (1) の解答欄中で、BNF の生成規則に自分で付けた番号 (IX, X, XI, XII, ...) から選んでもよい。（構文エラーの場合は、必ず ✗ を記入し、空欄のまま残さないこと。）

	if	**then**	**else**	**begin**	**end**	**write**	**id**	(	)	;	+	-	$
$S \rightarrow$													
$F \rightarrow$													
$L \rightarrow$													
$L' \rightarrow$													
$E \rightarrow$													
$E' \rightarrow$													

(6) この文法に対して、入力が文法にしたがっていれば「正しい構文です。」間違っていれば「構文に誤りがあります。」と表示する構文解析プログラムを作成する。プログラム（次ページ）中の指定の部分に入る S, F, E, E1, L, L1 関数のうち、F, L, L1 関数 の定義を完成させよ。ただし、S, F, E, E1, L, L1 は、それぞれ非終端記号 S, F, E, E', L, L' に対応する関数である。

（**プログラムの補足説明:** プログラム中では、終端記号は、“,” のような 1 文字のものは、その字そのもの（の ASCII コード）、**id** などのトークンは、C 言語のマクロ（例えば **id** の場合は ID）として表現している。入力の終わり（$）に対応するのは、このプログラムの場合、マクロ EOF である。

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を代入する。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。）

reportError 関数は、「構文に誤りがあります。」と表示し、 プログラムを終了する。

再帰下降構文解析プログラム

```
#include <stdio.h>    /* printf(), EOF など */
#include <stdlib.h>   /* exit() 用 */
#include <string.h>   /* strcmp() 用 */
#include <ctype.h>    /* isalpha() 用 */

/* 終端記号に対するマクロの定義 */
#define IF    257     /* トークン if */
#define THEN  258     /* トークン then */
#define ELSE  259     /* トークン else */
#define BEGIN 260     /* トークン begin */
#define END   261     /* トークン end */
#define WRITE 262     /* トークン write */
#define ID    263     /* トークン id */
int token;            /* 大域変数の宣言 */

/* 関数プロトタイプ宣言 */
void reportError(void);
int  yylex(void);
void eat(int t);

void S(void);
void F(void);
void E(void);
void E1(void);
```

```
void L(void);
void L1(void);


/* ************************************************************** */
/* この部分に 関数 S, F, E, E1, L, L1 の定義を挿入する。              */
/* ************************************************************** */

/* ここ以降は解答に直接関係はない。 */
void reportError(void) {
  printf("構文に誤りがあります。\n"); exit(0); /* プログラムを終了 */
}

int main() {   /* main関数 */
  token = yylex(); /* 最初のトークンを読む */
  S();
  if (token == EOF) {
    printf("正しい構文です!\n");
  } else {
    reportError();
  }
}

int yylex(void) {    /* 簡易字句解析ルーチン */
  int c;
  char buf[256];

  do { /* 空白は読み飛ばす。 */
    c = getchar();
  } while (c == '␣' || c == '\t' || c == '\n');

  if (isalpha(c)) {  /* アルファベットだったら… */
    char* ptr = buf;
    ungetc(c, stdin);
    while (1) {
      c = getchar();
      if (!isalpha(c) && !isdigit(c)) break;
      *ptr++ = c;
    }
    *ptr = '\0';
    ungetc(c, stdin);

    if (strcmp(buf, "if") == 0)    return IF;
    if (strcmp(buf, "then") == 0)  return THEN;
    if (strcmp(buf, "else") == 0)  return ELSE;
    if (strcmp(buf, "begin") == 0) return BEGIN;
    if (strcmp(buf, "end") == 0)   return END;
    if (strcmp(buf, "write") == 0) return WRITE;
    return ID;
  } 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) {
    /* 現在のトークンを捨てて、次のトークンを読む */
    token = yylex();
    return;
  } else {
    reportError();
  }
}
```

VI. （LR 構文解析）

次のような BNF で与えられる文法は曖昧であるが、優先順位・結合性を適切に指定することにより、LR 構文解析表を作成することができる。

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

ただし、

- ··· のあとの I, II などは生成規則の番号である。
- E は非終端記号、“(”, “)”, “*”, “|”, **a** は終端記号である。**a** はアルファベット 1 文字からなるトークンを表す。
- 開始記号（start symbol）は（当然）E である。

`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。）

	$	(	)	*	|	**a**	E
⓪		shift ②				shift ①	goto ③
①	reduce I						
②		shift ②				shift ①	goto ④
③	shift ⑤	shift ②		shift ⑦	shift ⑥	shift ①	goto ⑧
④		shift ②	shift ⑨	shift ⑦	shift ⑥	shift ①	goto ⑧
⑤	accept						
⑥		shift ②				shift ①	goto ⑩
⑦	reduce IV						
⑧	reduce III			shift ⑦	reduce III		goto ⑧
⑨	reduce V						
⑩	reduce II	shift ②	reduce II	shift ⑦	reduce II	shift ①	goto ⑧

注:
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。

次の入力列に対して、↑の次（右）の記号を**シフトした直後**の（つまりシフトしたあと、還元がまだ起こっていないときの）スタックの状態はどのようになっているか?

(1) **a a * | a ***　　(2) **a a a a * | a**　　(3) **a | a a | a ***

下の選択肢から選べ。（左がスタックの底とする）

(1) の選択肢

(A). ⓪ *E* ③ | ⑥

(B). ⓪ *E* ③ *E* ⑧ | ⑥

(C). ⓪ *E* ③ **a** ① * ⑦ | ⑥

(D). ⓪ *E* ③ *E* ⑧ * ⑦ | ⑥

(2) の選択肢

(A). ⓪ *E* ③ * ⑦

(B). ⓪ *E* ③ *E* ⑧ * ⑦

(C). ⓪ *E* ③ *E* ⑧ *E* ⑧ * ⑦

(D). ⓪ *E* ③ *E* ⑧ *E* ⑧ *E* ⑧ * ⑦

(3) の選択肢

(A). ⓪ *E* ③ * ⑦

(B). ⓪ *E* ③ | ⑥ *E* ⑩ * ⑦

(C). ⓪ *E* ③ | ⑥ *E* ⑩ | ⑥ *E* ⑩ * ⑦

(D). ⓪ *E* ③ | ⑥ *E* ⑩ *E* ⑧ | ⑥ *E* ⑩ * ⑦

コンパイラ・期末テスト計算用紙

（冊子から切り離しても良い）

コンパイラ・期末テスト計算用紙

（冊子から切り離しても良い）

コンパイラ（2017 年度）・期末テスト解答用紙（2017 年 08 月 03 日）

学籍番号		氏名	

I. （Backus-Naur 記法） (3×4)

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

II. （正規表現） (3×4)

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

III. （コンパイラのフェーズ） (2×4)

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

IV. （演算子順位法） (2×5)

(1)		(2)		(3)		(4)		(5)	

V. （再帰下降構文解析） (3, 3, 5, 4, 6, 5)

(1)	$E \rightarrow$ $E' \rightarrow$
(2)	{ }
(3)	{ }

裏ページに続く。

		if	**then**	**else**	**begin**	**end**	**write**	**id**	(	)	;	+	-	$
(4)	$L \rightarrow$													
	$L' \rightarrow$													
(5)	$S \rightarrow$													
	$F \rightarrow$													
	$E \rightarrow$													
	$E' \rightarrow$													

(6)

```
void F(void) {
  if (token == ID) {
                                          /* ←ここを埋める← */
  } else if (token == '(') {
                                          /* ←ここを埋める← */
  } else reportError();
}

void L(void) { /* ↓ここを埋める↓ */




}

void L1(void) { /* ↓ここを埋める↓ */




}
```

VI. （LR 構文解析） (4×3)

(1)		(2)		(3)	

授業・テストの感想

アンケート

無記名で解答して下さい。