コンパイラ（2013 年度）・期末テスト問題用紙

（2013 年 08 月 01 日（木）・ 10:30 〜 12:00）

（問題訂正適用済み）

訂正は赤字

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。

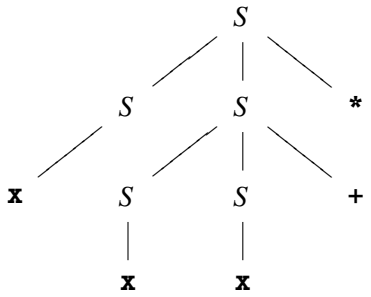
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$S \rightarrow S\ S\ \texttt{+} \mid S\ S\ \texttt{*} \mid \texttt{x}$$

ただし、S は非終端記号、“**+**”, “*****”, “**x**” は終端記号である。次の各記号列について、S 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: xxx+*に対する解析木



(1) `x x * x +`　　(2) `x + x x *`

(3) `x x + x x * *`　　(4) `x x x x * + +`

II. （正規表現）

以下の文字列について、
「`(ba*b)*b`」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、
「`(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 (R) を、
両方の正規表現に全体がマッチする文字列には (B) を、
どちらにも全体がマッチしない文字列には (N) をつけよ。

(1) `babaababb`　　(2) `bbbaaaabb`　　(3) `babbaabba`　　(4) `babbabbbb`

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) （for 文の「;」を「,」と書き間違えた。）

```
#include <stdio.h>
int main(void) {
    int i;
    for(i=0, i<10, i++) {
        printf("Hello!␣World\n");
    }
    return 0;
}
```

(2) （ポインタ型変数に浮動小数点数を代入しようとした。）

```
#include <stdio.h>
int main(void) {
    double *x;
    x = 3.14;
    printf("%f\n", x);
    return 0;
}
```

(3) （文字列リテラルの「"」を「'」と書き間違えた。）

```
#include <stdio.h>
int main(void) {
    printf('Hello!␣World\n');
    return 0;
}
```

(4) （最後の「}」を忘れた。）

```
#include <stdio.h>
int main(void) {
    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。

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

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。

「<」は非結合、「|」は左結合、「=」は右結合であり、「<」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`	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)`
`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 ⋖（シフト）、⋗（還元）、✗（エラー）のうちもっとも適切なもので埋めよ。

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。

$$
\begin{array}{lcll}
C & \rightarrow & \textbf{begin}\ L\ \textbf{end} & \cdots\ \text{I} \\
 & | & \textbf{stmt} & \cdots\ \text{II} \\
L & \rightarrow & L\ \text{“}\textbf{;}\text{”}\ C & \cdots\ \text{III} \\
 & | & C & \cdots\ \text{IV}
\end{array}
$$

ただし、「C」,「L」は非終端記号で、「**begin**」,「**stmt**」,「**end**」,「**;**」は終端記号とする。開始記号（start symbol）は C である。$\cdots$ の後の I, II などは生成規則の番号である。

(1) L の左再帰を除去せよ。補助的に新しく導入する非終端記号は L' とせよ。（後の (4) の解答で使用するために、生成規則に番号（V, VI, ...）を付けておいてもよい。）
以下の (2)〜(4) は、(1) で L から左再帰を除去して得られた BNF について答えよ。

(2) ***First***(C) を求めよ。

(3) ***Follow***(L') を求めよ。

(4) 下の構文解析表をすべて埋めよ。

	begin	**end**	**stmt**	**;**	**$**
$C \rightarrow$					
$L \rightarrow$					
$L' \rightarrow$					

(4) の解答は、上記の BNF 中の生成規則の番号（I 〜 II）、(1) の解答欄中で、BNF の生成規則に自分で付けた番号（V 〜 ）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。

(5) 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 C, L, L1 関数の定義を完成させよ。ただし、C, L, L1 は、それぞれ非終端記号 C, 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）、**begin** などのキーワードは、C 言語のマクロ（例えば **begin** の場合は `BEGIN`）として表現している。

`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。）
`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。

```
#include <stdio.h>
#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 BEGIN   257     /* トークン begin */
#define END     258     /* トークン end */
#define STMT    259     /* トークン stmt */

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 C(void);
void L(void);
void L1(void);

/* ************************************************************** */
 * この部分に 関数 C, 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 */
  C();
  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, "begin") == 0) {
      return BEGIN;
    } else if (strcmp(buf, "end") == 0) {
      return END;
    } else if (strcmp(buf, "stmt") == 0) {
      return STMT;
    } else {
      reportError();
    }
  } 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）

次のような文法

$$
\begin{array}{lcll}
S & \to & \text{“}\texttt{x}\text{”} & \cdots \ \text{I} \\
 & | & \text{“}\{\text{”}\ B\ \text{“}\}\text{”} & \cdots \ \text{II} \\
B & \to & B\ S & \cdots \ \text{III} \\
 & | & \varepsilon & \cdots \ \text{IV}
\end{array}
$$

に対して、LR 構文解析表を作成する。ただし、

- ··· の後の I, II などは生成規則の番号である。
- S, B は非終端記号、“{”, “x”, “}” は終端記号である。
- 開始記号は S 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 a 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。）

	$	x	{	}	S	B
⓪		shift ①	shift ②		goto ③	
①	reduce I					
②	reduce IV					goto ④
③	shift ⑤					
④		shift ①	shift ②	shift ⑥	goto ⑦	
⑤	accept					
⑥	reduce II					
⑦	reduce III					

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 番を使って還元」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) { x { } x }　　(2) { { x x x } }
　　↑　　↑

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 { ② B ④ x ①
(B). ⓪ { ② B ④ { ② } ⑥ x ①
(C). ⓪ { ② B ④ S ⑦ x ①
(D). ⓪ { ② B ④ { ② B ④ } ⑥ x ①

(2) の選択肢
(A). ⓪ { ② B ④ x ①
(B). ⓪ { ② B ④ { ② B ④ x ①
(C). ⓪ { ② B ④ { ② B ④ S ⑦ x ①
(D). ⓪ { ② B ④ S ⑦ { ② B ④ S ⑦ x ①

コンパイラ・期末テスト計算用紙

コンパイラ・期末テスト計算用紙

コンパイラ（2013 年度）・期末テスト解答用紙（2013 年 08 月 01 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） (3×4)

(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） (4, 4, 4, 6, 6)

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

裏ページに続く。

(4)

	begin	**end**	**stmt**	**;**	$
$C \rightarrow$					
$L \rightarrow$					
$L' \rightarrow$					

(5)

```
void C(void) {
  if (token == BEGIN) { /* ↓ここを埋める↓ */

  } else if (token == STMT) {
    eat(STMT);
  } 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） (5×2)

(1)		(2)	

授業・テストの感想