コンパイラ（2018年度）・期末テスト問題用紙

（2018年08月02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。

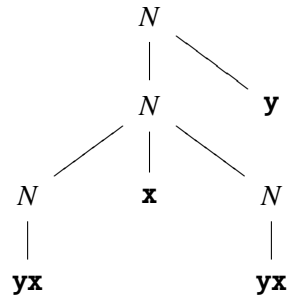
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$N \;\rightarrow\; N\,\mathtt{y} \;\mid\; N\,\mathtt{x}\,N \;\mid\; \mathtt{xy} \;\mid\; \mathtt{yx}$$

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



(1) **yxyyx**　(2) **xyxyxy**　(3) **yxxyxyy**　(4) **yxxxyxxy**

II. （正規表現）

以下の文字列について、
「(wx*w)*w」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、
「(w|xwwx)*」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) wxwxxwxww　(2) wxwwxwwww　(3) wxwwxxwwx　(4) wwwxxxxww

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!\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) （最後の「}」が二重になった。）

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

(4) （文字列リテラルの「"」を「'」で閉じようとした。）

```
#include <stdio.h>
int main(void) {
    printf("Hello!\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、「==」は非結合、「&&」は左結合、「>」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`	（構文エラー）
`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。

$$
\begin{array}{lcll}
C & \to & \textbf{begin}\ L\ \textbf{end} & \cdots\ \text{I} \\
 & | & \textbf{stmt} & \cdots\ \text{II} \\
 & | & \textbf{id}\ \text{“=”}\ E\ \text{“;”} & \cdots\ \text{III} \\
L & \to & L\ C & \\
 & | & C & \\
E & \to & E\ \text{“(”}\ E\ \text{“)”} & \\
 & | & E\ \text{“.”}\ \textbf{id} & \\
 & | & \textbf{id} & \\
 & | & \text{“(”}\ E\ \text{“)”} &
\end{array}
$$

ただし、「C」,「L」,「E」は非終端記号で、「**begin**」,「**end**」,「**stmt**」,「**id**」,「=」,「**;**」,「(」,「)」,「**.**」は終端記号とする。

開始記号（start symbol）は C である。また、$\cdots$ 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。

$$
\begin{array}{lcll}
L & \to & C\ L' & \cdots\ \text{IV} \\
L' & \to & \varepsilon & \cdots\ \text{V} \\
 & | & C\ L' & \cdots\ \text{VI}
\end{array}
$$

これを参考にして、E から左再帰を除去せよ。補助的に新しく導入する非終端記号は E' とせよ。（後の解答で使用するために、生成規則に番号（VII, VIII, ...）を付けておいてもよい。）

以下の (2)〜(4) は、(1) で L, E 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 $ で表すことにする。

(2) ***First***(C) を求めよ。

(3) ***Follow***(L') を求めよ。

(4) ***Follow***(E') を求めよ。

(5) 下の予測型構文解析表の C, L, L' の行を埋めよ。この問題の解答は ✗, I 〜 VI 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。）

(6) 下の予測型構文解析表の E, E' の行を埋めよ。

ただし、この解答は、✗ と (1) の解答欄中で、BNF の生成規則に自分で付けた番号 (VII, VIII, ...) 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。）

	begin	**end**	**stmt**	**id**	=	;	(	)	.	$
$C \rightarrow$										
$L \rightarrow$										
$L' \rightarrow$										
$E \rightarrow$										
$E' \rightarrow$										

(7) 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, E, E1 関数のうち、C, E, E1 関数 の定義を完成させよ。ただし、C, L, L1, E, E1 は、それぞれ非終端記号 C, L, L', E, E' に対応する関数である。予測型構文解析表の ✗ に相当する入力には reportError 関数を呼び出すようにすること。

（**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** などのトークンは、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。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 BEGIN   257     /* トークン begin */
#define END     258     /* トークン end */
#define STMT    259     /* トークン stmt */
#define ID      260     /* 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 C(void);
void L(void);
void L1(void);
void E(void);
void E1(void);
```

```
/* ************************************************************** */
 * この部分に 関数 C, L, L1, E, E1 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;
    if (strcmp(buf, "end") == 0)   return END;
    if (strcmp(buf, "stmt") == 0)  return STMT;
    if (strcmp(buf, "id") == 0)    return ID;
    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 BNF で与えられる文法

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

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 (start symbol) 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					

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) { { } x x }　(2) { { x { } x } }　(3) { { x { 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

(3) の選択肢
(A). ⓪ { ② { ② *B* ④ { ② *B* ④ } ⑥
(B). ⓪ { ② { ② x ① { ② x ① x ① } ⑥
(C). ⓪ { ② *B* ④ { ② *B* ④ { ② *B* ④ } ⑥
(D). ⓪ { ② *B* ④ { ② *B* ④ { ② *B* ④ x ① } ⑥

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（2018 年度）・期末テスト解答用紙（2018 年 08 月 02 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, 2, 3, 4, 3, 6, 5)

(1)	$E \rightarrow$ $E' \rightarrow$
(2)	{ }
(3)	{ }
(4)	{ }

裏ページに続く。

		begin	**end**	**stmt**	**id**	=	;	(	)	.	$
(5)	$C \rightarrow$										
	$L \rightarrow$										
	$L' \rightarrow$										
(6)	$E \rightarrow$										
	$E' \rightarrow$										

(6)

```
void C(void) {
  switch (token) {
  case BEGIN:                                   /* ←ここを埋める← */
  case STMT:                                    /* ←ここを埋める← */
  case ID:                                      /* ←ここを埋める← */
  default:  reportError();
}

void E(void) { /* ↓ここを埋める↓ */




}

void E1(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