コンパイラ（2019年度）・期末テスト問題用紙

（2019年08月01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。

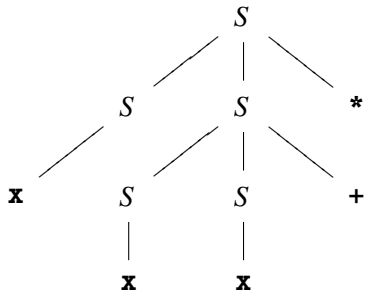
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

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

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、

「`(zw|wz)*(z|ε)`」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、

「`(zwz|wzw)*(w|ε)`」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) を記せ。

(1) `wzwzwzwzz`　　(2) `zwzwzwwzw`　　(3) `zwzwzwwzz`　　(4) `wzwzwzzwz`

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) （文末のセミコロン “;” を忘れた。）

```
#include <stdio.h>

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

(2) （文字列リテラルに二重引用符 「"」を忘れた。）

```
#include <stdio.h>

int main(void) {
    printf(Hello);
    return 0;
}
```

(3) （コメントを閉じるのを忘れた。）

```
#include <stdio.h>

int main(void) { /* 閉じ忘れのコメント
    printf("Hello World!\n");
    return 0;
}
```

(4) （括弧の種類を間違えた。）

```
#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。

$$E \rightarrow \quad \textbf{id} \;\mid\; E \text{ “\$” } E \;\mid\; E \text{ “>” } E \;\mid\; E \text{ “:” } E \;\mid\; E \text{ “+” } E \;\mid\; \text{“(” } E \text{ “)”}$$

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。

> 「+」は左結合、「:」は右結合、「>」は非結合、「$」は右結合、であり、「+」は「:」よりも優先順位が高く、「:」 は「>」よりも優先順位が高く、「>」 は「$」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`	`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)`

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)〜(8) 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)	(6)	⋗	⋖	⋗	⋖	⋗
$	(7)	⋖	⋖	(8)	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≐	⋖	✗
)	⋗	⋗	⋗	⋗	✗	⋗	✗	⋗
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。

$$
\begin{array}{lcl}
S & \rightarrow & S \text{ “;” } S \;\mid\; \texttt{id} \text{ “=” } E \;\mid\; \texttt{pr} \text{ “(” } E \text{ “)”} \\
E & \rightarrow & \texttt{id} \;\mid\; E \text{ “+” } E \;\mid\; \text{“(” } S \text{ “@” } E \text{ “)”}
\end{array}
$$

ただし、「S」,「E」は非終端記号で、「;」,「**id**」,「=」,「**pr**」,「(」,「)」,「+」,「@」は終端記号とする。開始記号（start symbol）は S である。

(1) E から左再帰を除去すると、次のような BNF が得られる。

$$
\begin{array}{lcll}
E & \rightarrow & \texttt{id}\, E' & \cdots \text{ I} \\
 & \mid & \text{“(” } S \text{ “@” } E \text{ “)” } E' & \cdots \text{ II} \\
E' & \rightarrow & \varepsilon & \cdots \text{ III} \\
 & \mid & \text{“+” } E\, E' & \cdots \text{ IV}
\end{array}
$$

ここで、$\cdots$ の後のローマ数字（I, II など）は生成規則の番号である。これを参考にして、S から左再帰を除去せよ。補助的に導入する非終端記号は S' とせよ。（後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 S と 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***(S') を求めよ。

(3) ***Follow***(S') を求めよ。

(4) ***Follow***(E') を求めよ。

(5) 下の予測型構文解析表の E, E' の行を埋めよ。この問題の解答は ✗, I 〜 IV の中から選べ。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 S, S' の行を埋めよ。

ただし、この問題の解答は、✗ と (1) の解答欄中で、BNF の生成規則に自分で付けた番号 (V, VI, ...) から選んでもよい。また、この文法は曖昧なため衝突が起こる可能性がある。その場合は、一つの欄に 2 つのの生成規則を I + II のように記入せよ。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。

	;	**id**	=	**pr**	(	)	+	@	$
$S \rightarrow$									
$S' \rightarrow$									
$E \rightarrow$									
$E' \rightarrow$									

(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 S, S1, E, E1 関数のうち、E, E1 関数 の定義を完成させよ。

ただし、S, S1, E, E1 は、それぞれ非終端記号 S, S', 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）、**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 ID   257      /* トークン id */
#define PR   258      /* トークン pr */

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 S1(void);
void E(void);
void E1(void);

/* ************************************************************** */
 * この部分に 関数 S, S1, 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 */
  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, "id") == 0) return ID;
    if (strcmp(buf, "pr") == 0) return PR;
    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）

「^」,「_」などの演算子はテキスト整形言語 LaTeX で使われている演算子で、`x^a` は上付きの添字 x^a、また `x_a` は下付きの添字 x_a を表す。LaTeX では `x_a^b` を特別扱いして、これを x_{a^b} や ${x_a}^b$ ではなく、x_a^b のように整形する。

このことを踏まえて...

次のような BNF で与えられる文法

$$
\begin{array}{rcll}
E & \to & E \text{ “\_” } E \text{ “\^{}” } E & \cdots I \\
 & | & E \text{ “\_” } E & \cdots II \\
 & | & E \text{ “\^{}” } E & \cdots III \\
 & | & \text{“\{” } E \text{ “\}”} & \cdots IV \\
 & | & \mathbf{a} & \cdots V
\end{array}
$$

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。
- E は非終端記号である。
- “_”,”^”, “{”, “}”, “**a**” は終端記号である。このうち、”**a**” はアルファベット 1 文字からなるトークンを表す。
- “^”, “_” 演算子の優先度は等しく、どちらも右結合である。

`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 ①		shift ②		goto ③
①			shift ①		shift ②		goto ④
②	reduce V						
③	shift ⑥	shift ⑦				shift ⑤	
④	shift ⑥	shift ⑦		shift ⑧			
⑤	accept						
⑥			shift ①		shift ②		goto ⑨
⑦			shift ①		shift ②		goto ⑩
⑧	reduce IV						
⑨	shift ⑥	shift ⑪	reduce II				
⑩	shift ⑥	shift ⑦	reduce III				
⑪			shift ①		shift ②		goto ⑫
⑫	shift ⑥	shift ⑦	? ? ? ? ? ?				

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。

(1) 〜 (2)

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^b_c^d`　　(2) `a^{b_c}^d`

下の選択肢（(1)〜(2) 共通）から選べ。（左がスタックの底とする）

(A) ⓪E③^⑦　(B) ⓪E③^⑥　(C) ⓪E③^⑥E⑨^⑪　(D) ⓪E③^⑦E⑩^⑦

(E) ⓪E③_⑥E⑨^⑪　(F) ⓪E③_⑦E⑩^⑦

(G) ⓪E③^⑦E⑩_⑥E⑨^⑪　(H) ⓪E③^⑥E⑨_⑪E⑨^⑪

(3) `a_b^c` という入力に対しては、`c` をシフトしたあと、まず生成規則 V による還元を行なって、DFA は⑫という状態になる。「還元還元衝突（reduce/reduce conflict）の時は、上（先）に書かれている構文規則が優先する。」という Bison の 衝突回避規則に従うと、LR 構文解析表の ？？？？？？ の部分には何が入るべきか、次の選択肢から選べ。

(A) reduce I　(B) reduce II　(C) reduce III　(D) reduce IV　(E) reduce V

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（2019年度）・期末テスト解答用紙（2019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） (2×4)

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

IV. （演算子順位法） (1×8)

(1)		(2)		(3)		(4)	
(5)		(6)		(7)		(8)	

裏ページに続く。

V. （再帰下降構文解析） (3, 2, 3, 3, 4, 5, 5)

(1)

$S \rightarrow$

$S' \rightarrow$

(2) { }

(3) { }

(4) { }

		;	**id**	=	**pr**	(	)	+	@	$
(5)	$E \rightarrow$									
	$E' \rightarrow$									
(6)	$S \rightarrow$									
	$S' \rightarrow$									

(7)

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

void E1(void) { /* ↓ここを埋める↓ */




}
```

VI. （LR 構文解析） (5×3)

(1)		(2)		(3)	

授業・テストの感想