コンパイラ（'11 年度）・期末テスト問題用紙

（'11 年 7 月 28 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. 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）

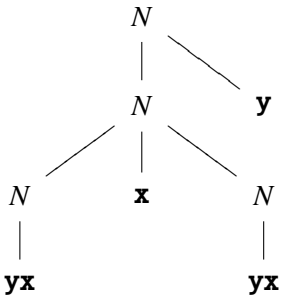
次のような BNF で表される文法を考える。

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

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。）

(1) **xyxyyxy**　(2) **yxxxyyy**　(3) **yxxxyxxy**

例: yxxyxy に対する解析木



II. （正規表現）

以下の文字列について、「**0(0|101)***」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)、「**0(10)*1**」という正規表現に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)、どちらにもマッチしない文字列には (N) をつけよ。

(1) **01010101**　(2) **010100101**　(3) **010101**　(4) **0101010**

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)〜(3) 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)〜(3) 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 文のセミコロン ; が一つ多い。）

```
#include <stdio.h>

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

(2) （文字列リテラルに *演算子を適用しようとした。）

```
#include <stdio.h>

int main(void) {
    printf("Hello!␣World\n" * 3);
    return 0;
}
```

(3) （コメントの終わりを示す「*/」を忘れた。）

```
#include <stdio.h>

int main(void) {
    printf("Hello!␣World\n");  /* 表示する *
    return 0;
}
```

(1)〜(3) 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（静的）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、まった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。

「:」は右結合、「>」は非結合、「>>」は左結合であり、「:」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`	`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)`
`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)〜(4) 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)	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	≐	⋖	✗
)	⋗	⋗	⋗	✗	⋗	✗	⋗
id	⋗	⋗	⋗	✗	⋗	✗	⋗

V. （再帰下降構文解析）

次のような BNF で表された文法に対して、再帰的下降構文解析ルーチンを作成する。

Statement	→	**id** “=” *Expr* “**;**”
	|	**if** “(” *Expr* “)” *Statement* **else** *Statement*
	|	**while** “(” *Expr* “)” *Statement*
	|	“{” *StatementList* “}”
StatementList	→	ε
	|	*Statement StatementList*
Expr	→	**num**
	|	*Expr* “**,**” **num**

ただし、*Statement*, *StatementList*, *Expr* は非終端記号、**id**, **if**, **else**, **while**, **num**, “=”, “(”, “)”, “**;**”, “{”, “}”, “**,**” は終端記号である。開始記号（start symbol）は *Statement* である。

(1) *Expr* から左再帰を除去せよ。補助的に導入する非終端記号は *Expr'* とせよ。
以下の (2)〜(4) は、(1) で *Expr* から左再帰を除去して得られた BNF について答えよ。

(2) ***Follow***(*StatementList*) を求めよ。

(3) ***Follow***(*Expr'*) を求めよ。

(4) 下の構文解析表の *StatementList* の行を埋めよ。

	id	=	**;**	**if**	(	)	**else**	**while**	{	}	**num**	**,**	$
Statement →													
StatementList →													
Expr →													
Expr' →													

(4) の解答は次の選択肢から選べ。
（構文エラーの場合は (G) を選択し、空欄のまま残さないこと。）

(A). **id** “=” *Expr* “**;**”
(B). **if** “(” *Expr* “)” *Statement* **else** *Statement*
(C). **while** “(” *Expr* “)” *Statement*
(D). “{” *StatementList* “}”
(E). ε
(F). *Statement StatementList*
(G). ✗

ただし、✗ は “構文誤り” を示す。

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、優先順位を適当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。

$$
\begin{array}{rcll}
E & \to & \textbf{while}\ E\ \textbf{do}\ E & \cdots\ \text{I} \\
 & | & \textbf{id}\ \text{“=”}\ E & \cdots\ \text{II} \\
 & | & E\ \text{“+”}\ E & \cdots\ \text{III} \\
 & | & \textbf{id} & \cdots\ \text{IV}
\end{array}
$$

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、**while**, **do**, **id**, “=”, “+” は終端記号である。**id** 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。）

	while	**do**	**id**	=	+	$	E
⓪	shift ①		shift ②				goto ③
①	shift ①		shift ②				goto ④
②	reduce IV			shift ⑤	reduce IV		
③					shift ⑦	shift ⑥	
④		shift ⑧			shift ⑦		
⑤	shift ①		shift ②				goto ⑨
⑥	accept						
⑦	shift ①		shift ②				goto ⑩
⑧	shift ①		shift ②				goto ⑪
⑨	reduce II				shift ⑦	reduce II	
⑩	reduce III						
⑪	reduce I				shift ⑦	reduce I	

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時の）スタックの状態はどのようになっているか?

(1) **id+id+id**　　(2) **while id+id do id=id+id**

次の選択肢から選べ。（左がスタックの底とする）

(1) の選択肢

(A). ⓪*E*③+⑦　　(B). ⓪**id**②+⑦**id**②+⑦

(C). ⓪**id**②+⑦*E*⑩+⑦　　(D). ⓪*E*③+⑦*E*⑩+⑦

(2) の選択肢

(A). ⓪*E*③+⑦　　(B). ⓪**while**①**id**②+⑦**id**②**do**⑧*E*⑪+⑦

(C). ⓪**while**①*E*④**do**⑧**id**②=⑤*E*⑨+⑦　　(D). ⓪**while**①*E*④**do**⑧*E*⑪+⑦

コンパイラ（’11 年度）・期末テスト解答用紙 (’11 年 7 月 28 日)

学籍番号		氏名	

I. （Backus-Naur 記法） (4×3)

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

II. （正規表現） (4×4)

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

III. （コンパイラのフェーズ） (4×3)

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

IV. （演算子順位法） (4×4)

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

裏ページに続く。

V. （再帰下降構文解析） (4×4)

(1).
Expr →

Expr' →

(2). { }

(3). { }

	id	=	**;**	**if**	(	)	**else**	**while**	{	}	**num**	**,**	$
(4). *StatementList* →													

VI. （LR 構文解析） (4×2)

(1).		(2).	

テストの感想