コンパイラ（2015年度）・期末テスト問題用紙

（2015年07月30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。

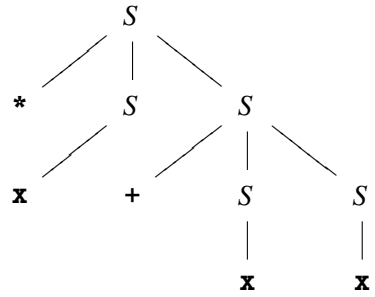
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$
\begin{aligned}
S \quad &\rightarrow \quad \texttt{+}\, S\ S \\
&\mid \quad \texttt{*}\, S\ S \\
&\mid \quad \texttt{x}
\end{aligned}
$$

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、上の BNF の非終端記号 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: `* x + x x` 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、
「**(yx|x(yz)*)***」という正規表現にマッチする 先頭からの 最長の部分文字列の文字数を答えよ。例えば、**yxxyzyzyyzxy** という文字列について考えると、その先頭の部分文字列 **yxxyzyz** は上の正規表現にマッチするが、それより長い先頭からの部分文字列: **yxxyzyzy**, **yxxyzyzyy**, . . . , **yxxyzyzyyzxy** はいずれもマッチしないので、マッチする先頭からの最長の部分文字列の文字数は 7 となる。（なお、文字列の途中からの部分文字列は考えなくて良い。）

(1) **xyzyzyxyzyxy**　　(2) **xyzyzyxxyzyz**
(3) **yxzxyzyxxyzx**　　(4) **xyzyxxyzyxxx**

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) （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;
}
```

(2) （文末のセミコロン “;” を忘れた。）

```
#include <stdio.h>

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

(3) （文字列リテラルに一重引用符 「'」を使った。）

```
#include <stdio.h>

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

(4) （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;
}
```

(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
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。

$$
\begin{aligned}
L &\rightarrow A \mid L \text{“!”} A \mid L \text{“\^{}”} A \\
A &\rightarrow N \mid A \text{“\&”} N \\
N &\rightarrow \text{“(”} L \text{“)”} \mid \textbf{id}
\end{aligned}
$$

ただし、「L」,「A」,「N」は非終端記号で、「!」,「^」,「&」,「**(**」,「**)**」,「**id**」は終端記号とする。開始記号（start symbol）は L である。

(1) A から左再帰を除去すると、次のような BNF が得られる。

$$
\begin{aligned}
A &\rightarrow N A' \\
A' &\rightarrow \varepsilon \mid \text{“\&”} N A'
\end{aligned}
$$

これを参考にして、L から左再帰を除去せよ。補助的に導入する非終端記号は L' とせよ。

以下の (2)〜(4) は、(1) で L と A から左再帰を除去して得られた BNF について答えよ。

(2) *Follow*(L') を求めよ。

(3) *Follow*(A') を求めよ。

(4) 下の構文解析表の A, A' の行を埋めよ。ただし、$ は入力の終わりを表す。

	id	(	)	!	^	&	$
$L \rightarrow$							
$L' \rightarrow$							
$A \rightarrow$							
$A' \rightarrow$							
$N \rightarrow$							

(4) の解答は次の選択肢から選べ。空欄のままにしないこと。

(A) $N A'$　(B) ε　(C) “&” $N A'$　(D) ✗

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) 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 `L`, `L1`, `A`, `A1`, `N` 関数のうち、`A`, `A1`, `N` 関数の定義を完成させよ。ただし、`L`, `L1`, `A`, `A1`, `N` は、それぞれ非終端記号 L, L', A, A', N 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。

`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>
#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 x */
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 L(void);
void L1(void);
void A(void);
void A1(void);
void N(void);

/* ************************************************************** */
 * この部分に 関数 L, L1, A, A1, N 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 */
  L();
  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);

    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）

次のような文法（··· の後の I, II などは生成規則の番号）

$$
\begin{array}{lcll}
E & \to & \textbf{id} & \cdots I \\
 & | & \text{“\{” } X \text{ “\}”} & \cdots II \\
 & | & E \text{ “\{” } X \text{ “\}”} & \cdots III \\
 & | & E \text{ “\#” } \textbf{id} & \cdots IV
\end{array}
\qquad
\begin{array}{lcll}
X & \to & \textbf{id} \text{ “=” } E & \cdots V \\
 & | & X \text{ “;” } \textbf{id} \text{ “=” } E & \cdots VI
\end{array}
$$

に対して、LR 構文解析表を作成する。ただし、

- 「E」,「X」は非終端記号で、「**id**」,「{」,「}」,「#」,「=」,「;」は終端記号とする。このうち、「**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 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。）

	id	{	}	#	=	;	$	E	X
⓪	shift ①	shift ②						goto ③	
①	reduce I								
②	shift ④								goto ⑤
③		shift ⑦		shift ⑧			shift ⑥		
④					shift ⑨				
⑤			shift ⑩			shift ⑪			
⑥	accept								
⑦	shift ④								goto ⑫
⑧	shift ⑬								
⑨	shift ①	shift ②						goto ⑭	
⑩	reduce II								
⑪	shift ⑮								
⑫			shift ⑯			shift ⑪			
⑬	reduce IV								
⑭	reduce V	shift ⑦	reduce V	shift ⑧	reduce V				
⑮					shift ⑰				
⑯	reduce III								
⑰	shift ①	shift ②						goto ⑱	
⑱	reduce VI	shift ⑦	reduce VI	shift ⑧	reduce VI				

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) `{b=c;s=t}{x=y}`　(2) `a{b=c#x}{s=t#y;u=v}`　(3) `{a={x=y;z=u}}#a`

下の選択肢（(1)〜(3) 共通）から選べ。（左がスタックの底とする）

(A). ⓪E③{⑦**id**④

(B). ⓪E③#⑧**id**⑬

(C). ⓪E③{⑦X⑫;⑪**id**⑮

(D). ⓪E③{⑦**id**④=⑨E⑭;⑪**id**⑮

(E). ⓪E③{⑦**id**④=⑨E⑭#⑧**id**⑬

(F). ⓪E③{⑦**id**④=⑨E⑭}⑯{⑦**id**④

(G). ⓪E③{⑦X⑫;⑪**id**④=⑨E⑭;⑪**id**⑮

(H). ⓪E③{⑦**id**④=⑨E⑭; ⑪**id**④=⑨E⑭;⑪**id**⑮

(I). ⓪E③{⑦**id**④=⑨E⑭}⑯{⑦**id**④=⑨E⑭}⑯{⑦**id**④

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（2015 年度）・期末テスト解答用紙（2015 年 07 月 30 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） (3, 4, 4, 6, 5)

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

裏ページに続く。

(4)

	id	(	)	!	^	&	$
$A \rightarrow$							
$A' \rightarrow$							

(5)

```
void A(void) { /* ↓ここを埋める↓ */




}

void A1(void) { /* ↓ここを埋める↓ */




}

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

VI. （LR 構文解析） (4×3)

(1)		(2)		(3)	

授業・テストの感想