# コンパイラ（2016年度）・期末テスト問題用紙

（2016年07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. 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。

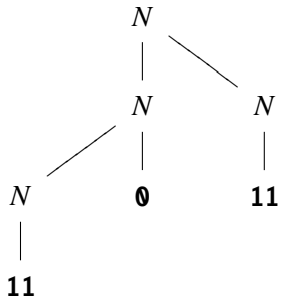
I. （Backus-Naur 記法）

次のような BNF で表される文法を考える。

$$
\begin{aligned}
N \quad &\rightarrow \quad N\,\texttt{0} \\
&\mid \quad N\,N \\
&\mid \quad \texttt{11} \\
&\mid \quad \texttt{1001}
\end{aligned}
$$

ただし、$N$ は非終端記号、“**0**”, “**1**” 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 $N$ 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:** 11011 に対する解析木




(1) `1100100`　(2) `1111011`　(3) `1110010`　(4) `1001011`

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) を、「**`0(10)*1`**」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)、両方に全体がマッチする文字列には (B)、どちらにも全体がマッチしない文字列には (N) を記せ。

(1) **`010110101`**　(2) **`01010101`**　(3) **`010101011`**　(4) **`010100101`**

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) （コメントの終わりを示す「*/」を忘れた。）

```
#include <stdio.h>

int main(void) {
    printf("Hello! World\n");  /* ハローと表示する
    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) （文末のセミコロン; の位置を間違えた。）

```
#include <stdio.h>

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

(4) （for 文のセミコロン; をコンマ, と間違えた。）

```
#include <stdio.h>

int main(void) {
    int i;
    for (i = 0, i < 5, i++) {
        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)〜(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。

$$
\begin{aligned}
E &\rightarrow \textbf{id} \mid E \text{ “(” } E \text{ “)”} \mid \text{“\{” } E \text{ “|” } L \text{ “\}”} \\
L &\rightarrow L \text{ “,” } S \mid S \\
S &\rightarrow \textbf{id} \text{ “:” } E
\end{aligned}
$$

ただし、「$E$」,「$L$」,「$S$」は非終端記号で、「**id**」,「**(**」,「**)**」,「**{**」,「**|**」,「**}**」,「**,**」,「**:**」は終端記号とする。開始記号（start symbol）は $E$ 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。

$$
\begin{aligned}
L &\rightarrow S\ L' \\
L' &\rightarrow \varepsilon \mid \text{“,” } S\ L'
\end{aligned}
$$

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

以下の (2)〜(4) は、(1) で $E$ と $L$ 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 $ で表せ。

(2) *Follow*($L'$) を求めよ。

(3) *Follow*($E'$) を求めよ。

(4) 下の予測型構文解析表の $L$, $L'$ の行を埋めよ。この問題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) $S\ L'$　　(B) $\varepsilon$　　(C) “,” $S\ L'$　　(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) 下の予測型構文解析表の $E$, $E'$ の行を埋めよ。（この問題はヒントとなる選択肢はない。ただし、“エラー” の欄は明示的に ✗ を埋め、空欄のままにしないこと。）

| | **id** | ( | ) | { | \| | } | , | : | $ |
|---|---|---|---|---|---|---|---|---|---|
| $E \rightarrow$ | | | | | | | | | |
| $E' \rightarrow$ | | | | | | | | | |
| $L \rightarrow$ | | | | | | | | | |
| $L' \rightarrow$ | | | | | | | | | |
| $S \rightarrow$ | | | | | | | | | |

(6) 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 E, E1, L, L1, S 関数のうち、E, E1, S 関数の定義を完成させよ。ただし、E, E1, L, L1, S は、それぞれ非終端記号 $E$, $E'$, $L$, $L'$, $S$ 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。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 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 E(void);
void E1(void);
void L(void);
void L1(void);
void S(void);

/* ********************************************************************* */
/* この部分に 関数 E, E1, L, L1, S 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 */
  E();
  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 BNF で与えられる文法は曖昧であるが、優先順位・結合性を適切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 | |

**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) **id = id = id**　　(2) **id = id + id + id**　　(3) **id + while id do id = id + id**

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*E*③=⑤　　(B). ⓪**id**②=⑤**id**②=⑤

(C). ⓪**id**②=⑤*E*⑨=⑤　　(D). ⓪*E*③=⑤**id**②=⑤

(2) の選択肢

(A). ⓪**id**②=⑤**id**②+⑦**id**②+⑦　　(B). ⓪**id**②=⑤*E*⑨+⑦*E*⑨+⑦

(C). ⓪**id**②=⑤*E*⑨+⑦　　(D). ⓪*E*③=⑤*E*⑨+⑦

(3) の選択肢

(A). ⓪*E*③+⑦**while**①*E*④ **do**⑧**id**②=⑤*E*⑨+⑦

(B). ⓪*E*③+⑦

(C). ⓪*E*③+⑦**while**①*E*④ **do**⑧*E*⑪+⑦

(D). ⓪*E*③+⑦*E*⑩+⑦

このページは空白です。

# 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（2016 年度）・期末テスト解答用紙（2016 年 07 月 28 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, 3, 5, 4, 6, 5)

| | |
|---|---|
| (1) | $E \rightarrow$<br><br>$E' \rightarrow$ |
| (2) | { } |
| (3) | { } |

裏ページに続く。

| | | **id** | ( | ) | { | \| | } | , | : | $ |
|---|---|---|---|---|---|---|---|---|---|---|
| (4) | $L \rightarrow$ | | | | | | | | | |
| | $L' \rightarrow$ | | | | | | | | | |
| (5) | $E \rightarrow$ | | | | | | | | | |
| | $E' \rightarrow$ | | | | | | | | | |

(6)

```
void E(void) { /* ↓ここを埋める↓ */




}

void E1(void) { /* ↓ここを埋める↓ */




}

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

VI. （LR 構文解析） (4×3)

| (1) | | (2) | | (3) | |
|---|---|---|---|---|---|

授業・テストの感想