システムプログラム・テスト問題用紙

('00年 2月 15日（火）・ 10:30 〜 12:00)

解答上、その他の注意事項

I. 問題は、問 I 〜 V までである。
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. 選択式でない問で解答欄がマス目になっている場合は、1字に 1マスを用いること。特に空白にも必ず 1マスを用いること
V. 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。
VI. ノート・プリント・参考書などは持ち込み可である。
VII. テストの配点は 62点である。（第1回レポート 18点・第2回レポート 20点）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）

下の記号列の中で、次の BNF

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

の 非終端記号 S から生成されるものには○、生成されないものには×をつけよ。

(1) $xx + x*$

(2) $xxx+$

(3) $x * x + x$

(4) $xxx + *$

II.（コンパイラの構造・語句）

次のコンパイラの構造に関する文章の、(1) 〜 (6) の空欄に当てはまる語句を、下の選択肢から選べ。

コンパイラの処理は、いくつかのフェーズ（処理過程）に分割できる。まず入力（ソースプログラム）は [(1)] で、いくつかのトークン（意味の最小単位）に分けられる。C 言語で言えば、`if` や`while` などのキーワードや、変数名、関数名などの識別子、数や文字列等の定数、`{`, `}`, `;` などの区切り文字がトークンである。[(1)] は、[(2)] などのプログラムで正規表現から自動生成するか、手書きで作成する。

次にトークンの列は、[(3)] で、構文木という形に構造化される。[(3)] は、[(4)] などのプログラムで BNF などから自動生成するか、演算子順位法や再帰的下向き構文解析法などの手法を用いて、手書きで作成される。

その次に、[(5)] で型のチェックなどを行ない、これまでのフェーズで発見できないプログラムのエラーをチェックする。

この後、最適化などの処理を行ない、最後は [(6)] でアセンブリ言語や機械語などのターゲットプログラムを出力する。

(2) と (4) の選択肢

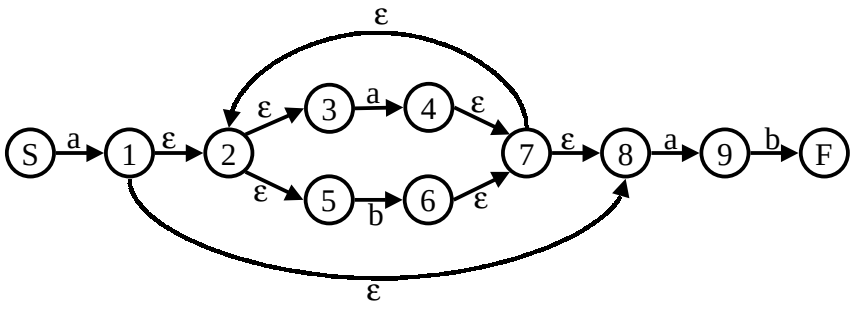
(A). `bcc32`　(B). `dir`　(C). `lex`（または `flex`）　(D). `yacc`（または `bison`）

(1), (3), (5), (6) の選択肢

(A). 意味解析部　(B). 構文解析部　(C). コード生成部　(D). 字句解析部

（(6) の解答は最後に残ったものになるので、答案用紙に解答欄はない。）

III.（正規



上のような NFA（非決定性有限オートマトン）がある。 ただし S は開始状態、F は終了状態である。これと同じ記号列を受理する DFA（決定性有限オートマトン）を部分集合構成法により作成せよ。（作成した DFA の状態が、もとの NFA の状態のどの集合に対応するかをわかるようにすること。）

IV.（演算子順位法）

次の文法は、（C 言語の）論理式を表している。

$E \rightarrow$ **id**
| E “`&&`” E
| E “`||`” E
| “`!`” E
| “`(`” E “`)`”

ただし、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。

> 「`&&`」と「`||`」は、いずれも左結合で、「`!`」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、`!a&&b` は `(!a)&&b` と解釈され、`a || b && c` は `a || (b && c)` と解釈される。

以下の演算子順位行列の空欄を ⋖ , ≐ , ⋗ , **err** のいずれかで埋めよ。（**err** はエラーを表す。）

左 \ 右	`||`	`&&`	`!`	`(`	`)`	**id**	終
始	⋖	⋖	⋖	⋖	**err**	⋖	≐
`||`	(1)	⋖	⋖	⋖	⋗	⋖	⋗
`&&`	⋗	⋗	⋖	⋖	⋗	⋖	⋗
`!`	⋗	(2)	**err**	⋖	⋗	⋖	⋗
`(`	⋖	⋖	⋖	(3)	≐	⋖	**err**
`)`	⋗	⋗	**err**	**err**	⋗	**err**	⋗
id	⋗	⋗	**err**	**err**	⋗	**err**	⋗

V.（再帰的下向き構文解析）

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

$$
\begin{array}{lcl}
\textit{Statement} & \rightarrow & \textbf{id}\ \text{“=”}\ \textit{Expr}\ \text{“;”} \\
& | & \textbf{if}\ \text{“(”}\ \textit{Expr}\ \text{“)”}\ \textit{Statement}\ \textbf{else}\ \textit{Statement} \\
& | & \textbf{while}\ \text{“(”}\ \textit{Expr}\ \text{“)”}\ \textit{Statement} \\
& | & \text{“\{”}\ \textit{StatementList}\ \text{“\}”} \\
\textit{StatementList} & \rightarrow & \epsilon \\
& | & \textit{Statement}\ \textit{StatementList} \\
\textit{Expr} & \rightarrow & \textbf{num}
\end{array}
$$

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。

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

(2) 〜 (4)

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。プログラム中の空欄を埋めよ。

（**プログラムの補足説明:** プログラム中では、終端記号は、“{”, “}”, “;” のように一字からなるものは、その字そのもの（の ASCII コード）、**if**, **else** などは、C 言語のマクロとして 256 以上の整数値で表現されている。ただし、**if**, **else** などは、すべて C 言語のキーワードなので、これらの終端記号を表すマクロの名前として、`IF`, `ELSE`, ...のように、綴りをすべて大文字にしたものを用いている。

`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が代入される。`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。）

(5) 関数 `StatementList()` の定義を再帰呼出しを用いない形に書き換えよ。`for` 文もしくは `while` 文を用い、`goto` 文は用いないこと。

また、解答欄を節約するために、

「`token == ID || token == IF || token == WHILE || token == '}'`」という条件式は Ⓐ、「`token == '}'`」という条件式は Ⓑ と略記して良い。

```
/*
    最初の部分（#include 文、マクロの定義、yylex(), eat() の定義など）は、
    問題に関係ないので省略
 */

void Statement() {
  if (token == ID) {
     eat(ID); eat('='); [      (2)      ]; eat(';');
  } else if (token == IF) {
    eat(IF); eat('('); Expr(); eat(')'); Statement(); eat(ELSE); Statement();
  } else if (token == WHILE) {
    eat(WHILE); eat('('); Expr(); eat(')'); [      (3)      ];
  } else if (token == '{') {
    eat('{'); [      (4)      ];  eat('}');
  } else {
    printf("構文に誤りがあります。¥n"); exit(0); /* プログラムを終了 */
  }
}

void StatementList() {
  if (token == ID || token == IF || token == WHILE || token == '{') {
    Statement(); StatementList();
  } else if (token == '}') {
    /* 何もしない */
  } else {
    printf("構文に誤りがあります。¥n"); exit(0); /* プログラムを終了 */
  }
}

void Expr() {
  if (token == NUM) {
    eat(NUM);
  } else {
    printf("構文に誤りがあります。¥n"); exit(0); /* プログラムを終了 */
  }
}

/* main 関数 */
int main() {
  token = yylex(); /* 最初のトークンを読む */
  Statement();
  printf("正しい構文です!¥n");
}
```

（参考） 上のプログラムで省略された最初の部分（問題には直接は関係ない）

```
#include <stdio.h>
#include <stdlib.h>   /* exit() 用 */
#include <ctype.h>    /* isdigit() 用 */

/* 終端記号に対するマクロ */
#define ID      256     /* トークン id */
#define IF      257     /* トークン if */
#define ELSE    258     /* トークン else */
#define WHILE   259     /* トークン while */
#define NUM     260     /* トークン num */

/* 大域変数の宣言 */
int token;

int yylex() {   /* 簡易字句解析ルーチン */
  int c;
  char buf[256];

  do {
    c = getchar();
  } while (c == ' ' || c == '¥t' || c == '¥n');

  if (isalpha(c)) {
    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, "if")==0) {
      return IF;
    } else if (strcmp(buf, "else")==0) {
      return ELSE;
    } else if (strcmp(buf, "while")==0) {
      return WHILE;
    } else {
      return ID;
    }
  } else if (isdigit (c)) {
    double poi;
    ungetc(c, stdin); scanf("%lf", &poi);
    return NUM;
  } 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 {
    printf("構文に誤りがあります。");
    exit (0); /* プログラムを終了 */
  }
}

/* 関数プロトタイプ宣言 */
void Statement(void);
void StatementList(void);
void Expr(void);
```

システムプログラム・テスト解答用紙 ('00 年 2 月 15 日)

学籍番号		氏名	

I. (3×4)

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

II. (2×5)

(1).		(2).		(3).		(4).		(5).	

III. (8)

IV. (4×3)

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

V. (3×4+8)

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

(5) の解答欄は裏面

(5).

テストの感想