2018 年 11 月 12 日

コンパイラ及び演習
中間試験

日程

- 2018 年 11 月 12 日(月)，第 8 回
- 2018 年 11 月 26 日(日) 23:59，　レポート及びプログラムの提出締切

レポート執筆における注意点

- グループワーク可（数人程度まで）
  ただし，共同作業者の学籍番号と氏名をレポートの表紙に記載すること．

解答・レポート作成に関して

- ポータルサイトからレポートの表紙をダウンロードして使用すること．表紙以外のフォーマットは自由．
- レポートは A4 縦で作成すること．
- 共同作業者の学籍番号と氏名をレポートの表紙に記載すること．
- 参考文献やウェブサイトを参照にした際には，レポートの最終ページに明記すること．

提出物とその詳細

提出物：　レポートとソースコード，演習課題(実装)

提出方法と提出期限：

- レポートとソースコード
  - 提出方法と提出先：ポータルサイトの「課題管理」
  - レポートとソースコードを 1 つの ZIP ファイルとする
  - 提出期限：2018 年 11 月 26 日(日) 23:59
- 演習課題(実装)
  - 提出方法：演習課題提出システム(講義と同様)
  - 提出期限：2018 年 11 月 26 日(月) 0:00

提出物の詳細：

- レポート（A4 縦）
  - ファイルフォーマットは PDF とする
  - 指定の表紙を付けること
  - 関数の一覧．各関数の引数・戻り値・機能について簡単に記すこと．
    講義で配布したサンプルコードの関数を変更せずに用いる場合，これらの関数は一覧に載せる必要はない．
  - 参考文献
- ソースコード

- ファイル名は compiler-mid.c とすること．
  (採点の都合により，ファイル名は必ず守ること)
- C 言語で記述すること．
- gcc でコンパイルできること．
  - 実装が完成せずに提出する場合でも，コンパイルは通るようにしてください．コンパイルが通らない場合，実装の点数は大幅に減点されます．
- レポートの関数一覧とソースコード内の関数が対応していること．

(次ページに続く)

問 1

後述の字句の定義・文法は，変数を使用可能な四則演算(電卓)を規定する．この文法について考える．

問 1.1

コンパイラが処理可能な記号列(プログラム)は文法によって導出される記号列である．例えば，記号列 `"a = 123"` は次の通り導出される．なお，空白文字は文法に含まれていないが，読みやすさのために必要に応じて空白文字を入れている．

<program> ⇒ <statement> ⇒ VARIABLE '=' <expression> ⇒ a '=' <expression> ⇒ a '=' <term> ⇒ a '=' <factor> ⇒ a '=' VARIABLE ⇒ a '=' 123

本試験で与えられる文法は，記号列`"a = (123 + 45) ?a"` を導出できることを示せ．上記同様，必要に応じて空白文字を入れてよいとする．

問 1.2

本試験で与えられる文法は，記号列`"a = 123 + 45 + 678"` を導出できない．文法を参照し，この記号列を導出できない理由を述べよ．

問 2

字句の定義・文法で構成されるコンパイラを実装せよ．実装方針は講義の演習と原則同じとする．実装における条件等は後述の「実装の条件」や「仕様と文法の補足」などを参照すること．

問 3

本課題で与えられている文法では剰余演算が定義されていない．剰余演算の記号を `'%'` として，剰余演算を本問の BNF で与えられた定義に追加するとき，文法をどのように修正すればよいか述べよ（実装する必要はない）．追加にあたっては，演算子の優先順位を考慮すること．

(次ページに続く)

課題

変数を使用可能な四則演算(電卓)

電卓の機能

- 入出力
  - 入力：式が記述されたテキストファイル
  - 出力：式の演算結果を標準出力に出力
- 整数の四則演算が可能
  - 括弧による演算の優先順位付けを含む
- 変数が使用可能

実装の条件

- 後述の文法を満たす
- 字句解析を行なう
- 構文解析は下降型構文解析で実装する

補足 : 上記の条件を満たしていれば, 講義で配布したサンプルコードは使用していなくとも構いません.

字句の定義

- VARIABLE は，'_', 'a'-'z', 'A'-'_Z' からなる記号列であり，変数を意味する.
- NUMBER は，'0'-'9' からなる記号列であり，10 進数の数値を意味する.

BNF 記法による文法

```
〈program〉     ::= 〈statement〉 | 〈statement〉〈program〉
〈statement〉   ::= VARIABLE '=' 〈expression〉
                 | '?' 〈expression〉
〈expression〉  ::= 〈term〉
                 | 〈term〉 '+' 〈term〉
                 | 〈term〉 '-' 〈term〉
〈term〉        ::= 〈factor〉
                 | 〈factor〉 '*' 〈factor〉
                 | 〈factor〉 '/' 〈factor〉
〈factor〉      ::= VARIABLE | NUMBER
                 | '(' 〈expression〉 ')'
```

(次ページに続く)

仕様と文法の補足

- 入力
  - 式（記号列）はテキストファイルから読み込む
- 文法の補足
  - 変数
    - VARIABLE で表現される変数が使用可能
    - 変数の宣言はない．初出の変数に値を代入した時点から使用可能とする
    - 変数への代入（再代入を含む）が可能
  - 定数
    - NUMBER で表現される
  - 演算子
    - '+', '-', '*', '/', '(', ')'の意味と優先順位は，通常の四則演算に従う．
      ただし，除算'/'の演算結果は C 言語の int 同士の演算に従い，小数点以下切り捨てとする．
  - 代入
    - "VARIABLE = <expression>" の形式で，式<expression>の演算と代入が可能である．式の演算結果は VARIABLE に格納される．
  - 式の演算結果の出力
    - "'?' <expression>" は，<expression>の演算結果を標準出力に出力する．演算結果を x としたとき，出力は次のフォーマットで出力されるとする．
      printf("result: %d¥n", x);
    - 注意 1：本課題の演算結果は整数のみ
    - 注意 2：1 つの'?'に対して，1 つの出力がされる
- 出力
  - 標準出力に対する "'?' <expression>" の結果
- エラー処理とエラー出力
  - "error: " に続けてエラーメッセージを表示する．
  - スタックのエラー
    - オーバーフロー："stack overflow"
      - 例: "error: stack overflow"
    - アンダーフロー："stack underflow"
      - 例: "error: stack underflow"
  - 算術エラー
    - 0 による除算："division by zero"
      - 例: "1 / 0"に対して，"error: division by zero "
  - 文法エラー
    - '='がない："'=' is expected"
      - 変数のみの"a"だけの記述に対して，"error: '=' is expected "
    - ')'がない："')' is expected"

- "error: ')' is expected"

✧ 右括弧')'に対応する左括弧'('がない：")"

- "error: ')'"

✧ '?' に続く<expression>がない："<expression> is expected"

- "error: <expression> is expected"

✧ 文法から導出できない記号が現れた場合："(その記号)"

- 例: "a = 1 + 2 - 3"に対して，"error: -"

✧ 未定義の記号："(未定義の記号)"

- 例: "4 % 2"に対して，"error: % "

✧ 記号表にない記号が使われた場合（一度も値を代入されていない記号が使われた場合）：""%s" is not in the symbol table"

- 例: 変数 b に一度も値が代入されていなとき"a = b"に対して，"error: "b" is not in the symbol table "

以上