Forritunarmál
Skilaverkefni 2: Þýðandi og túlkur

Þetta forritunarverkefni tengist efni sem farið hefur verið yfir í fyrstu þremur vikum námskeiðsins, þ.e. um *sýndarvélar* (e. abstract machines), *lýsingu forritunarmála* (e. description of programming languages) og *lesgreiningu og þáttun* (e. lexical analysis and parsing). Þið eigið að útfæra forritin í verkefninu í **Python**.

Lýsing (e. description)

Forritunarverkefnið ykkar er tvíþætt:

1) Skrifið þýðanda sem þýðir forrit í einföldu forritunarmáli `E` (E stendur fyrir *expressions*) yfir á milliþulu `S` sem byggir á stafla (e. stack-based intermediate code). Milliþulan `S` keyrir í sýndarvél sem útfærð er í hluta 2. Strengur í málinu `E` samanstendur af lista af setningum, þar sem sérhver setning er annað hvort gildisveitingarsetning (e. assignment statement) eða print setning.

   Samhengisfrjáls mállýsing (e. context-free grammar) `G` fyrir `E` er:

   Statements -> Statement **;** Statements | **end**
   Statement -> **id** **=** Expr | **print id**
   Expr- > Term | Term **+** Expr | Term **–** Expr
   Term -> Factor | Factor ***** Term
   Factor -> **int** | **id** | **(** Expr **)**

   Máleiningar (e. non-terminals) eru: *Statements* (byrjunareining), *Statement*, *Expr*, *Term*, og *Factor*. Tókar (e. tokens/terminals) eru:
   ; end id = print + - * int ( )

   Milliþulan `S` samanstendur af eftirfarandi skipunum:

```
PUSH op     // pushes the operand op onto the stack
ADD         // pops the two top elements from the stack, adds their values
            // and pushes the result back onto the stack
SUB         // pops the two top elements from the stack, subtracts the first
            // value retrieved from the second value,
            // and pushes the result back onto the stack
MULT        // pops the two top elements from the stack, multiplies their
            // values and pushes the result back onto the stack
ASSIGN      // pops the two top elements from the stack, assigns the first
            // element (a value) to the second element (a variable)
PRINT       // prints the value currently on top of the stack
```

2) Útfærið sýndarvél fyrir `S`, þ.e. túlk (e. interpreter) sem gerir kleift að keyra kóða sem skrifaður er í `S`.

Útfærslan (e. implementation)

Farið eftir eftirfarandi leiðbeiningum í útfærslunni ykkar (skref 1-4 eiga við þýðandann og skref 5 á við túlkinn).

1. Útfærið klasann **EToken** sem inniheldur bæði les (e. lexeme) í formi strengs og tókakóða (e. token code) í formi integer fasta:

   Integer fastarnir eru eftirfarandi:
   { ID, ASSIGN, SEMICOL, INT, PLUS, MINUS, MULT, LPAREN, RPAREN, PRINT, END, ERROR}

   (Sjá skýring á fastanum ERROR í lið 2).

   Útfærið þessa fasta inni í klasanum EToken þannig að hægt sé að vísa í þá með t.d. `EToken.ID`.

2. Útfærið klasann **ELexer** sem er lesgreinir. Hann þarf að eiga sér *public* aðgerð (e. method), *get_next_token()*, sem les **standard input** (stdin) og leitar að mynstrum (e. patterns) sem passa við einhvern tóka. Notið **sys.stdin.read(1)** til að lesa inntakið staf fyrir staf.

   Athugið að les fyrir tókana PLUS, MINUS, MULT, LPAREN, RPAREN, ASSIGN, SEMICOL innihalda aðeins einn staf. Mynstur fyrir les fyrir aðra tóka eru:

   INT = [0-9]+
   ID = [A-Za-z]+
   END = end
   PRINT = print

   Lesgreinirinn skilar tókanum **ERROR** ef eitthvað ólöglegt les finnst.

   Þegar þið eruð búin að útfæra ELexer klasann þá getið þið prófað hann með forritinu *test_elexer.py* sem fylgir lýsingunni á verkefninu:

   ```
   python test_elexer.py < program.e
   ```
   (þar sem program.e er forrit í E)

3. Útfærið klasann **EParser** sem er þáttari. Þáttarinn á að vera útfærður sem *ofansækinn endurkvæmisþáttari* (e. recursive-descent parser) fyrir mállýsinguna `G` að ofan. Úttakið úr þáttaranum er milliþulan `S` sem skrifuð er út á **standard output**

(stdout). Ein auð lína skal vera í lok milliþulunnar. Þið þurfið að finna út á hvaða stað í þáttaranum skrifa þarf út einstakar milliþuluskipanir.

Ef inntaksforritið samrýmist ekki mállýsingunni (eða ef lesgreinirinn skilar ERROR tóka) þá skal þáttarinn skrifa út "Syntax error" (á þeim stað þar sem villan uppgötvast) og hætta keyrslu.

Þáttarinn skal hafa a.m.k. tvær tilvikabreytur (e. instance variables), önnur af taginu **ELexer** og hin af taginu **EToken** (fyrir núverandi tóka). Þáttarinn skal jafnframt hafa fallið, *parse()* sem setur þáttunina í gang, og *next_token()* sem sækir næsta tóka frá lesgreininum og skrifar út villu ef þarf:

```
def parse(self):
    self.next_token()
    self.statements()

def next_token(self):
    self.curr_token = self.lexer.get_next_token()
    if self.curr_token == EToken.ERROR:  # lexical error
        self.error()
```

Þáttarinn er síðan auðvitað með ýmis önnur föll sem endurspegla mállýsinguna.

4. Útfærið aðalforritið **ecompiler.py** sem gerir aðeins eftirfarandi:

```
lexer = ELexer()
parser = EParser(lexer)
parser.parse()
```

5. Útfærið klasann **SInterpreter** sem er túlkur. `sinterpreter.py` er jafnframt með main fall sem gerir eftirfarandi:

```
interpreter = SInterpreter()
interpreter.cycle()
```

cycle() framkvæmir *fetch-decode-execute cycle*, þ.e. les milliþuluna *S* línu fyrir línu úr **standard input** (stdin), afkóðar sérhverja skipun og kallar á aðgerðir í `SInterpreter` til að keyra viðkomandi skipun. Úttakið er skrifað út á **standard output**.

Ef `SInterpreter` rekst á óleyfilegan virkja (e. invalid operator) eða það eru ekki nógu margir þolendur (e. operands) fyrir virkjann á staflanum þá skal hann skrifa út villuboðin "Error for operator: nameOfOperator" (þar sem nameOfOperator er viðkomandi virki) og hætta keyrslu.

Notið *stafla* í `SInterpreter` til að vinna með milliþuluna (getið einfaldlega notað *list* í Python fyrir stafla) og notið *uppflettitöflu* (e. dictionary) til að geyma gildin á breytum.

Keyrsla/Prófun

Inntakið í **ecompiler.py** er forrit sem skrifað er í málinu `E`. Úttakið er samsvarandi milliþula `S` skrifuð út á standard output. Gerum t.d. ráð fyrir því að eftirfarandi forrit, skrifað í `E`, sé í skránni `program.e`

```
var = 3;
b = 4 * (7-var);
print b;
end
```

Þá keyrir eftirfarandi skipun þýðandann með inntaksforritinu:

```
python ecompiler.py < program.e
```

og eftirfarandi milliþula er skrifuð út á standard output:

```
PUSH var
PUSH 3
ASSIGN
PUSH b
PUSH 4
PUSH 7
PUSH var
SUB
MULT
ASSIGN
PUSH b
PRINT
```

<- Hér er auð lína

Úttakinu úr `ecompiler.py` getur auðvitað verið "redirected" í skrá og síðan túlkað af Interpreter:

```
python ecompiler.py < program.e > program.s

python sinterpreter.py < program.s

16
```

Þið þurfið líka að sjá til þess að hægt sé að keyra forritin ykkar á eftirfarandi hátt:

```
python ecompiler.py < program.e | python sinterpreter.py
```

`ecompiler.py` les frá standard input og skrifa út á standard output. Síðan les `sinterpreter.py` (frá standard input) úttakið sem `ecompiler.py` myndar og skrifar eigið úttak út á standard output. Þið eigið að búa til skeljarskriftu (e. shell script) `run` sem keyrir þetta ferli að ofan þannig að forritari geti keyrt forrit í `E` með einni skipun:

```
./run program.e
16
```

Dæmi um ólögleg forrit í E:

1) program.e:

```
var = 3 + ;
print var;
end

python ecompiler.py < program.e
PUSH var
PUSH 3
Syntax error
```

2) program.e:

```
var = 3 ! ;
print var;
end

python ecompiler.py < program.e
PUSH var
PUSH 3
Syntax error
```

Hverju á að skila

- Öllum forritum í einni .zip skrá: etoken.py, elexer.py, eparser.py, ecompiler.py, sinterpreter.py og run.