P0 Scanner Tests

Emil Sekerinski, McMaster University, revised February 2022

Tests scanner output and produces all scanner error messages.

In [ ]:
```
import nbimporter; nbimporter.options["only_defs"] = False
import SC
```

Procedure `scan` collects the symbols recognized by the scanner into a list. The list consists of pair with `SC.sym` and `SC.newline`; other variables of `SC` like `SC.val`, `SC.pos`, etc. are not included, but the code can easily be modified.

In [ ]:
```
from SC import TIMES, DIV, MOD, AND, PLUS, MINUS, OR, EQ, NE, LT, GT, \
    LE, GE, PERIOD, COMMA, COLON, NOT, LPAREN, RPAREN, LBRAK, RBRAK, \
    LARROW, RARROW, LBRACE, RBRACE, CARD, COMPLEMENT, UNION, INTERSECTION, \
    ELEMENT, SUBSET, SUPERSET, DOTDOT, THEN, DO, BECOMES, NUMBER, IDENT, \
    SEMICOLON, ELSE, IF, WHILE, CONST, TYPE, VAR, SET, PROCEDURE, PROGRAM, \
    INDENT, DEDENT, EOF

symbol = {IDENT: 'IDENT', NUMBER: 'NUMBER', TIMES: 'TIMES', DIV: 'DIV', MOD: 'MOD',
    PLUS: 'PLUS', MINUS: 'MINUS', AND: 'AND', OR: 'OR', EQ: 'EQ', NE: 'NE',
    LT: 'LT', GT: 'GT', LE: 'LE', GE: 'GE', SEMICOLON: 'SEMICOLON', COMMA: 'COMMA',
    COLON: 'COLON', BECOMES: 'BECOMES', PERIOD: 'PERIOD', DOTDOT: 'DOTDOT',
    NOT: 'NOT', LPAREN: 'LPAREN', RPAREN: 'RPAREN', LBRAK: 'LBRAK', RBRAK: 'RBRAK',
    LBRACE: 'LBRACE', RBRACE: 'RBRACE', LARROW: 'LARROW', RARROW: 'RARROW', CARD: 'CARD',
    COMPLEMENT: 'COMPLEMENT', UNION: 'UNION', INTERSECTION: 'INTERSECTION',
    ELEMENT: 'ELEMENT', SUBSET: 'SUBSET', SUPERSET: 'SUPERSET', IF: 'IF', THEN: 'THEN',
    ELSE: 'ELSE', WHILE: 'WHILE', DO: 'DO', CONST: 'CONST', TYPE: 'TYPE', VAR: 'VAR',
    SET: 'SET', PROCEDURE: 'PROCEDURE', PROGRAM: 'PROGRAM', INDENT: 'INDENT',
    DEDENT: 'DEDENT', EOF: 'EOF'}

def scan(src): # for a more readable scanner output
    SC.init(src); syms = []
    while SC.sym != SC.EOF:
        syms.append((symbol[SC.sym], SC.newline))
        SC.getSym()
    return syms
```

In [ ]:
```
assert scan("""

program p

  if a then
    writeln()
  else
    writeln()
  if a then writeln() else writeln()
""") == \
[('PROGRAM', True),
 ('IDENT', False),
 ('INDENT', False),
 ('IF', True),
 ('IDENT', False),
 ('THEN', False),
 ('INDENT', False),
 ('IDENT', True),
 ('LPAREN', False),
 ('RPAREN', False),
 ('DEDENT', False),
 ('ELSE', True),
 ('INDENT', True),
 ('IDENT', True),
 ('LPAREN', False),
 ('RPAREN', False),
 ('DEDENT', False),
 ('IF', True),
 ('IDENT', False),
 ('THEN', False),
 ('IDENT', False),
 ('LPAREN', False),
 ('RPAREN', False),
 ('ELSE', False),
 ('IDENT', False),
 ('LPAREN', False),
 ('RPAREN', False),
 ('DEDENT', False)]
```

In [ ]:
```
assert scan("""
type T = [1..10] → integer
```

```
var a: T
procedure r()
    a[3] := 9
program p
  a[3] := 9
""") == \
[('TYPE', True),
 ('IDENT', False),
 ('EQ', False),
 ('LBRAK', False),
 ('NUMBER', False),
 ('DOTDOT', False),
 ('NUMBER', False),
 ('RBRAK', False),
 ('RARROW', False),
 ('IDENT', False),
 ('VAR', True),
 ('IDENT', False),
 ('COLON', False),
 ('IDENT', False),
 ('PROCEDURE', True),
 ('IDENT', False),
 ('LPAREN', False),
 ('RPAREN', False),
 ('INDENT', False),
 ('IDENT', True),
 ('LBRAK', False),
 ('NUMBER', False),
 ('RBRAK', False),
 ('BECOMES', False),
 ('NUMBER', False),
 ('DEDENT', False),
 ('PROGRAM', True),
 ('IDENT', False),
 ('INDENT', False),
 ('IDENT', True),
 ('LBRAK', False),
 ('NUMBER', False),
 ('RBRAK', False),
 ('BECOMES', False),
 ('NUMBER', False),
 ('DEDENT', False)]
```

In [ ]:
```
assert scan("""
program
y := 5
if a then
  if b then
    a := b
x := 3
""") == \
[('PROGRAM', True),
 ('IDENT', True),
 ('BECOMES', False),
 ('NUMBER', False),
 ('IF', True),
 ('IDENT', False),
 ('THEN', False),
 ('INDENT', False),
 ('IF', True),
 ('IDENT', False),
 ('THEN', False),
 ('INDENT', False),
 ('IDENT', True),
 ('BECOMES', False),
 ('IDENT', False),
 ('DEDENT', False),
 ('DEDENT', False),
 ('IDENT', True),
 ('BECOMES', False),
 ('NUMBER', False)]
```

In [ ]:
```
assert scan("""
program p
  while 2 > 3 do
    write(1)
""") == \
[('PROGRAM', True),
 ('IDENT', False),
 ('INDENT', False),
 ('WHILE', True),
 ('NUMBER', False),
 ('GT', False),
```

```
 ('NUMBER', False),
 ('DO', False),
 ('INDENT', False),
 ('IDENT', True),
 ('LPAREN', False),
 ('NUMBER', False),
 ('RPAREN', False),
 ('DEDENT', False),
 ('DEDENT', False)]
```

In [ ]:
```
assert scan("""// comment 1
program p      // comment 2
  writeln()    // comment 3
               // comment 4
""") == \
[('PROGRAM', True),
 ('IDENT', False),
 ('INDENT', False),
 ('IDENT', True),
 ('LPAREN', False),
 ('RPAREN', False),
 ('DEDENT', False)]
```

Procedure `scanerr(s)` returns an empty string if scanning `s` succeeds or the error message produced while scanning; the error message is also printed.

In [ ]:
```
def scanerr(s):
    try: scan(s); return ''
    except Exception as e:
        print(e); return str(e)
```

In [ ]:
```
assert "number too large" in scanerr("""
const c = 12345678901234567890
""")
```

In [ ]:
```
assert "illegal character" in scanerr("""
program p_
  writeln()
""")
```