P0 Parsing Tests

Emil Sekerinski, McMaster University, revised February 2022

In [ ]:
```
import nbimporter; nbimporter.options["only_defs"] = False
from P0 import compileString
```

Procedure `compileerr(s)` returns an empty string if compiling `s` succeeds or the error message produced while compiling; the error message is also printed. The procedure is used here to test parsing.

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

Error "] expected"

In [ ]:
```
assert "] expected" in compileerr("""
var a: [1..10] → integer
var x: integer
program p
    x := a[4
""")
```

Error ") expected"

In [ ]:
```
assert ") expected" in compileerr("""
program p
  var x: integer
    x := (5
""")
```

Error "expression expected"

In [ ]:
```
assert "expression expected" in compileerr("""
program p
  var x: integer
    x := +
""")
```

Error "new line expected"

In [ ]:
```
assert "new line expected" in compileerr("""
program p
  write(5) write(7)
""")
```

Error "dedent or new line expected"

In [ ]:
```
assert "dedent or new line expected" in compileerr("""
program p
  write(5)
    write(7)
""")
```

Error "indented statement expected"

In [ ]:
```
assert "indented statement expected" in compileerr("""
procedure p()
  if 3 > 4 then
writeln()
""")
```

Error "variable for result expected"

In [ ]:
```
assert "variable for result expected" in compileerr("""
program p
  read()
""")
```

Error ":= or ← expected"

In [ ]:
```
assert ":= or ← expected" in compileerr("""
```

```
var a: [5 .. 7] → integer
program p
  var b: boolean
    a[5] +
""")
```

Error "'(' expected"

In [ ]:
```
assert "'(' expected" in compileerr("""
program p
  writeln
""")
```

Error "')' expected"

In [ ]:
```
assert "')' expected" in compileerr("""
program p
  writeln(
""")
```

Error "'then' expected"

In [ ]:
```
assert "'then' expected" in compileerr("""
program p
  if true write(5)
""")
```

Error "'do' expected"

In [ ]:
```
assert "'do' expected" in compileerr("""
program p
  while true write(5)
""")
```

Error "statement expected"

In [ ]:
```
assert "statement expected" in compileerr("""
program p
  write(3); const c = 5
""")
```

Error "'..' expected"

In [ ]:
```
assert "'..' expected" in compileerr("""
var a: [5 → integer
program p
  writeln()
""")
```

Error "']' expected"

In [ ]:
```
assert "']' expected" in compileerr("""
var a: [5..7 → integer
program p
  writeln()
""")
```

Error "'→' expected"

In [ ]:
```
assert "'→' expected" in compileerr("""
var a: [3 .. 7] integer
program p
  writeln()
""")
```

Error "type expected"

In [ ]:
```
assert "type expected" in compileerr("""
program p
  var x: if
""")
```

Error "identifier expected"

In [ ]:
```
assert "identifier expected" in compileerr("""
```

```
program p
  var if: integer
""")
```

Error "identifier expected"

In [ ]:
```
assert "identifier expected" in compileerr("""
program p
  var if: integer
""")
```

Error "identifier expected"

In [ ]:
```
assert "identifier expected" in compileerr("""
program p
  var x, if: integer
""")
```

Error "':' expected"

In [ ]:
```
assert "':' expected" in compileerr("""
program p
  var x integer
""")
```

Error "identifier expected"

In [ ]:
```
assert "identifier expected" in compileerr("""
program p
  var i, j: integer, if: boolean
""")
```

Error "identifier expected"

In [ ]:
```
assert "identifier expected" in compileerr("""
program p
  var i, j: integer, b, if: boolean
""")
```

Error "constant name expected"

In [ ]:
```
assert "constant name expected" in compileerr("""
program p
  const 5 = 7
  write(3)
""")
```

Error "= expected"

In [ ]:
```
assert "= expected" in compileerr("""
program p
  const c: 5
  write(5)
""")
```

Error "type name expected"

In [ ]:
```
assert "type name expected" in compileerr("""
program p
  type 5 = integer
  write(3)
""")
```

Error "= expected"

In [ ]:
```
assert "= expected" in compileerr("""
program p
  type T: integer
  writeln()
""")
```

Error "procedure name expected"

In [ ]:
```
assert "procedure name expected" in compileerr("""
procedure
```

```
  writeln()
program p
  writeln()
""")
```

Error "(expected"

In [ ]:
```
assert "( expected" in compileerr("""
procedure q
  writeln()
program p
  writeln()
""")
```

Error ") expected"

In [ ]:
```
assert ") expected" in compileerr("""
procedure q(
  writeln()
program p
  writeln()
""")
```

Error "(expected"

In [ ]:
```
assert "( expected" in compileerr("""
procedure q(x: integer) → boolean
  writeln()
program p
  writeln()
""")
```

Error ") expected"

In [ ]:
```
assert ") expected" in compileerr("""
procedure q(x: integer) → (y: boolean
  writeln()
program p
  writeln()
""")
```

Error "indent expected"

In [ ]:
```
assert "indent expected" in compileerr("""
program p
writeln()
""")
```

Error "dedent or new line expected"

In [ ]:
```
assert "dedent or new line expected" in compileerr("""
program p
  const c = 5
    writeln()
      writeln()
""")
```

Error "statement expected"

In [ ]:
```
assert "statement expected" in compileerr("""
program p
  program q
""")
```

Error "dedent or new line expected"

In [ ]:
```
assert "dedent or new line expected" in compileerr("""
program p
  writeln()
    writeln()
""")
```

Error "'program' expected"

In [ ]:
```
assert "'program' expected" in compileerr("""
var x: integer
```

```
""")
```

Error "program name expected"

In [ ]:
```
assert "program name expected" in compileerr("""
program
  writeln()
""")
```

Multiple Indentations

In [ ]:
```
assert compileString("""
procedure q()
  var b: boolean
    b := true
    if b then write(3)
    else write(5)
    if ¬b then write(5)
    else if b then write(7)
    else write(9)
    while b do
      if b then
        b := false; write(1)
program p
  if 3 > 4 then writeln() else
    q()
""") == """\
(module
(import "P0lib" "write" (func $write (param i32)))
(import "P0lib" "writeln" (func $writeln))
(import "P0lib" "read" (func $read (result i32)))
(func $q
(local $b i32)
(local $0 i32)
i32.const 1
local.set $b
local.get $b
if
i32.const 3
call $write
else
i32.const 5
call $write
end
local.get $b
i32.eqz
if
i32.const 5
call $write
else
local.get $b
if
i32.const 7
call $write
else
i32.const 9
call $write
end
end
loop
local.get $b
if
local.get $b
if
i32.const 0
local.set $b
i32.const 1
call $write
end
br 1
end
end
)
(global $_memsize (mut i32) i32.const 0)
(func $program
(local $0 i32)
i32.const 0
if
call $writeln
else
call $q
end
```

```
)
(memory 1)
(start $program)
)"""
```