SC

March 12, 2018

0.1 P0 Scanner

Original Author: Emil Sekerinski, February 2017 The scanner reads the characters of the source consecutively and recognizes symbols they form: - procedure `init(src)` initializes the scanner - procedure `getSym()` recognizes the next symbol and assigns it to variables `sym` and `val`. - procedure `mark(msg)` prints an error message at the current location in the source.

Symbols are encoded by integer constants.

```
In [2]: TIMES = 1; DIV = 2; MOD = 3; AND = 4; PLUS = 5; MINUS = 6
        OR = 7; EQ = 8; NE = 9; LT = 10; GT = 11; LE = 12; GE = 13
        PERIOD = 14; COMMA = 15; COLON = 16; RPAREN = 17; RBRAK = 18
        OF = 19; THEN = 20; DO = 21; LPAREN = 22; LBRAK = 23; NOT = 24
        BECOMES = 25; NUMBER = 26; IDENT = 27; SEMICOLON = 28
        END = 29; ELSE = 30; IF = 31; WHILE = 32; ARRAY = 33
        RECORD = 34; CONST = 35; TYPE = 36; VAR = 37; PROCEDURE = 38
        BEGIN = 39; PROGRAM = 40; EOF = 41
```

Following variables determine the state of the scanner: - `(line, pos)` is the location of the current symbol in source - `(lastline, lastpos)` is used to more accurately report errors - `(errline, errpos)` is used to suppress multiple errors at the same location - `ch` is the current character - `sym` the current symbol - if `sym` is `NUMBER`, `val` is the value of the number - if `sym` is `IDENT`, `val` is the identifier string - `source` is the string with the source program

The source is specified as a parameter to the procedure `init`:

```
In [ ]: def init(src):
            global line, lastline, errline, pos, lastpos, errpos
            global sym, val, error, source, index
            line, lastline, errline = 1, 1, 1
            pos, lastpos, errpos = 0, 0, 0
            sym, val, error, source, index = None, None, False, src, 0
            getChar(); getSym()
```

Procedure `getChar()` assigns the next character in `ch`, or assigns `chr(0)` at the end of the source. Variables `line`, `pos` are updated with the current location in the source and `lastline`, `lastpos` are updated with the location of the previously read character.

```
In [ ]: def getChar():
            global line, lastline, pos, lastpos, ch, source, index
            if index == len(source): ch = chr(0)
```

```
            else:
                ch, index = source[index], index + 1
                lastpos = pos
                if ch == '\n':
                    pos, line = 0, line + 1
                else:
                    lastline, pos = line, pos + 1
```

Procedure `mark(msg)` prints an error message with the current location in the source. To avoid a cascade of errors, only one error message at a source location is printed.

```
In [ ]: def mark(msg):
            global errline, errpos, error
            if lastline > errline or lastpos > errpos:
                print('error: line', lastline, 'pos', lastpos, msg)
            errline, errpos, error = lastline, lastpos, True
```

Procedure `number()` parses

```
number ::= digit {digit}
digit ::= '0' | ... | '9'
```

If the number fits in 32 bits, sets `sym` to `NUMBER` and assigns to number to `val`, otherwise reports an error.

```
In [ ]: def number():
            global sym, val
            sym, val = NUMBER, 0
            while '0' <= ch <= '9':
                val = 10 * val + int(ch)
                getChar()
            if val >= 2**31:
                mark('number too large'); val = 0
```

Procedure `identKW()` parses

```
identKW ::= identifier | keyword
identifier ::= letter {letter | digit}
letter ::= 'A' | ... | 'Z' | 'a' | ... | 'z'
keyword ::= 'div' | 'mod' | 'and' | 'or' | 'of' | 'then' |
            'do' | 'not' | 'end' | 'else' | 'if' | 'while' |
            'array' | 'record' | 'const' | 'type' |
            'var' | 'procedure' | 'begin' | 'program'
```

The longest sequence of character that matches `letter {letter | digit}` is read. If that sequence is a keyword, `sym` is set accordingly, otherwise `sym` is set to `IDENT`.

```
In [9]: KEYWORDS = \
            {'div': DIV, 'mod': MOD, 'and': AND, 'or': OR, 'of': OF, 'then': THEN,
```

```
            'do': DO, 'not': NOT, 'end': END, 'else': ELSE, 'if': IF, 'while': WHILE,
            'array': ARRAY, 'record': RECORD, 'const': CONST, 'type': TYPE,
            'var': VAR, 'procedure': PROCEDURE, 'begin': BEGIN, 'program': PROGRAM}

        def identKW():
            global sym, val
            start = index - 1
            while ('A' <= ch <= 'Z') or ('a' <= ch <= 'z') or \
                  ('0' <= ch <= '9'): getChar()
            val = source[start:index-1]
            sym = KEYWORDS[val] if val in KEYWORDS else IDENT
```

Procedure `comment()` parses

```
comment = '{' {character} '}'
```

If a comment is not terminated, an error is reported, otherwise the comment is skipped.

```
In [ ]: def comment():
            while chr(0) != ch != '}': getChar()
            if ch == chr(0): mark('comment not terminated')
            else: getChar()
```

Procedure `getSym()` parses

```
symbol ::=
    {blank} (identKW | number | comment | '*' | '+' | '-' | '=' |
    '<' | '<=' | '>' | '>=' | ';' | ',' | ':' | '=' | '.' | '(' |
    ')' | '[' | ']')
blank ::= chr(0) | ... | " "
```

If a valid symbol is recognized, `sym` is set accordingly, otherwise an error is reported. The longest match is used for recognizing operators. Blanks are skipped. At the end of the source, `sym` is set to `EOF`.

```
In [1]: def getSym():
            global sym
            while chr(0) < ch <= ' ': getChar()
            if 'A' <= ch <= 'Z' or 'a' <= ch <= 'z': identKW()
            elif '0' <= ch <= '9': number()
            elif ch == '{': comment(); getSym()
            elif ch == '*': getChar(); sym = TIMES
            elif ch == '+': getChar(); sym = PLUS
            elif ch == '-': getChar(); sym = MINUS
            elif ch == '=': getChar(); sym = EQ
            elif ch == '<':
                getChar()
                if ch == '=': getChar(); sym = LE
                elif ch == '>': getChar(); sym = NE
```

```
            else: sym = LT
        elif ch == '>':
            getChar()
            if ch == '=': getChar(); sym = GE
            else: sym = GT
        elif ch == ';': getChar(); sym = SEMICOLON
        elif ch == ',': getChar(); sym = COMMA
        elif ch == ':':
            getChar()
            if ch == '=': getChar(); sym = BECOMES
            else: sym = COLON
        elif ch == '.': getChar(); sym = PERIOD
        elif ch == '(': getChar(); sym = LPAREN
        elif ch == ')': getChar(); sym = RPAREN
        elif ch == '[': getChar(); sym = LBRAK
        elif ch == ']': getChar(); sym = RBRAK
        elif ch == chr(0): sym = EOF
        else: mark('illegal character'); getChar(); sym = None
```

0.2 Appendix

```
In [16]: %%HTML
         <style>
         div.prompt {display:none}
         </style>
```

```
<IPython.core.display.HTML object>
```