CS 375: Lexical Analyzer

Due: February 1, 2018.

Write a lexical analyzer for Pascal. The program may be written in C or in Lisp.

Input to the Lexical Analyzer is obtained by calling functions that get input characters. `getchar()` returns the next character from the input and moves the character pointer; `peekchar()` returns the next character without moving the pointer. `peek2char()` returns the second character without moving the pointer. These functions are provided.

The Lexical Analyzer is called as the function `gettoken()` (provided in the file `scanner.c`); its output is one token. A token record contains the following fields (plus some pointers, to be used later):

- `tokentype` is an integer (`0..5`) (in Lisp, a symbol) denoting the type of the token: `OPERATOR DELIMITER RESERVED IDENTIFIERTOK STRINGTOK NUMBERTOK .`
- `datatype` is an integer (in Lisp, a symbol) that denotes the type of basic data: `INTEGER REAL STRINGTYPE BOOLETYPE POINTER .`
- `tokenval` contains the value of the token. For C, use the subfields `whichval`, `stringval`, `intval`, and `realval`; the storage for these is overlapped, since only one of them will be used in a given kind of token.

White Space and Comments:

Blanks, tabs, ends of lines, and comments are considered to be separators; the Lexical Analyzer consumes (skips over) these, but does not return anything for them. Comments are contained between the characters `{` and `}` , or between `(*` and `*)` . The first occurrence of the terminating character(s) ends the comment; comments cannot be nested.

Operators and Delimiters:

Operators are as follows: `+ - * / := = <> < <= >= > ^ .`

The following Reserved Words are treated as Operators: `and or not div mod in`

Delimiters are: `, ; : ( ) [ ] ..`

The result returned for an Operator or Delimiter is:

`tokentype`:	`OPERATOR` (`0`) or `DELIMITER` (`1`)
`whichval`:	integer denoting which operator (`1..19`) or delimiter (`1..8`).

Reserved words:

The Reserved Words of Pascal (other than Operators) are:

```
array       downto     function   of           repeat     until
begin       else       goto       packed       set        var
case        end        if         procedure    then       while
const       file       label      program      to         with
do          for        nil        record       type
```

The result returned for a Reserved Word is:

`tokentype`: `RESERVED` (`2`)

`whichval`: integer denoting which reserved word (`1..29`)

Identifiers:

Identifiers begin with a letter, followed by any number of letters or digits. We will assume that only the first 15 characters of the identifier name are significant; longer identifiers should be truncated to 15 characters. We will assume that the case of letters is left as specified; our input will use lower-case letters.

The result returned for an Identifier is:
`tokentype`: `IDENTIFIERTOK` (`3`)
`stringval`: string (identifier name)

Strings:

Strings are enclosed by `'` characters. The character `'` may be included within a string by doubling it, as in the example `'don''t'` . For simplicity, we will assume that a string has at most 15 characters.

The result returned for a String is:
`tokentype`: `STRINGTOK` (`4`)
`stringval`: string

Unsigned Numbers:

Unsigned Numbers must begin with a digit; if there is a decimal point, it must be followed by at least one digit. A number may be followed by a signed decimal exponent, in which case it is floating-point (whether there is a decimal point or not).

The lexical analyzer must convert a number to internal numeric form. Care must be taken to ensure that the result is numerically accurate and that errors such as numeric overflow are detected. Challenging examples are included in the test data; correct handling of these examples will be a grading criterion. If there is an error, your program should print an error message and return a number token of the correct type (integer or real); the value of the number will not matter.

The result returned for a Number is:
`tokentype`: `NUMBERTOK` (`5`)
`datatype`: integer denoting type: `INTEGER` or `REAL`
`tokenval`: numeric value in internal form. For C, use `intval` and `realval`

Testing:

Test your program on the files `graph1.pas` and `scantst.pas` .

Reference:

Jensen, K. and Wirth, N., *Pascal User Manual and Report*, Springer-Verlag, 1976.

Notes for C:

Use the `makefile` in the directory `/projects/cs375/`. A file `lex1.c` containing program stubs for this assignment is provided; you may use this file as the starting point for your program. Use `make lex1` to compile this program:

```
mkdir test
cd test
cp /projects/cs375/* .
make lex1
lex1
123 45678 12
17 444
^C              to quit
```

Copy `lex1.c` to `lexanc.c` and extend `lexanc.c` for this assignment; use `make lexanc` to compile your program. The file `lexandr.c` contains a driver program to call `gettoken()` and print the results. The file `scanner.c` contains functions `peekchar` and `peek2char` and some utility functions. The function `getchar` is part of the `stdio.h` package. Use the definitions in `token.h` and `lexan.h`, which definine the `TOKEN` structure, operator numbers etc. The file `printtoken.c` contains the function `talloc`, which makes a new token data structure and returns a pointer to it. You probably should not change the files `scanner.c` and `printtoken.c`. Note that the tables in `printtoken.c` are intended for printing and may not be what you need for the lexical analyzer; copy and modify these tables to make new ones if you wish.

Note that a string in C is an array of characters terminated by a zero `'\0'` character; you should terminate all strings this way. A string cannot just be assigned as in Java; it must be copied in a loop one character at a time, or you can use `strcpy(to,from)`.

You must perform scanning at the character level; you are *not* allowed to use C library functions that do the work of the lexical analyzer, such as `sscanf`. However, you may use standard string library functions such as `strcmp` (string compare).

Notes for Lisp:

You should use Lisp only if you are already good at Lisp. However, there is little risk in doing the first project in Lisp.

The file `scanner.lsp` contains the functions `getchar`, `peekchar`, and `peek2char`; these return `NIL` at end-of-file. `charclass` determines the class of a character. Call the function `read-file` to read the input file, e.g.

```
(read-file "/u/cs375/graph1.pas")
```

After calling `read-file`, the call `(test-scanner)` will test your program on the file. The file `tokendefs.lsp` contains macro definitions for tokens and the functions `talloc` (which makes a token structure) and `printtoken`.

Character constants are written with a `#\` prefix, e.g. the character `*` is written `#\*` ; the blank space is written `#\Space` .

Useful Common Lisp functions: `char-code`, `make-string`, `char-downcase`, `char=`, `string=`.

Other useful files: `number.lsp  while.lsp`

You must perform scanning at the character level; you are *not* allowed to use Lisp functions that do the work of the lexical analyzer, such as `intern` or `read-from-string`.