

CS 405/505 ASSIGNMENT #4

Due Monday, September 26, 2005

Consider the following extended BNF grammar for a subset of COBOL (Common Business-Oriented Language), called MicroCOBOL.

cobol-source-program	::=	IDENTIFICATION DIVISION . PROGRAM-ID program-name [DATA DIVISION . WORKING-STORAGE SECTION . { data-description-entry }] PROCEDURE DIVISION . { sentence } STOP RUN . [{ nested-cobol-source-program } END PROGRAM program-name .]
nested-cobol-source-program	::=	IDENTIFICATION DIVISION . PROGRAM-ID program-name IS COMMON PROGRAM [DATA DIVISION . WORKING-STORAGE SECTION . { data-description-entry } [LINKAGE SECTION . { data-description-entry }]] PROCEDURE DIVISION [USING data-name { , data-name }] . { sentence } EXIT PROGRAM . END PROGRAM program-name .
data-description-entry	::=	level-number [data-name] data-description-entry-clauses .
data-description-entry-clauses	::=	[PIC S9(9)] [OCCURS integer TIMES]
sentence	::=	{ statement . }
statement	::=	call-statement compute-statement display-statement if-statement perform-statement
call-statement	::=	CALL identifier USING BY CONTENT identifier { , identifier } BY REFERENCE identifier
compute-statement	::=	COMPUTE identifier = arithmetic-expression
display-statement	::=	DISPLAY identifier
if-statement	::=	IF condition [THEN] { statement } [ELSE { statement }] [END-IF]
perform-statement	::=	PERFORM UNTIL condition { statement } END-PERFORM
condition	::=	combinable-condition { logical-operator combinable-condition }
combinable-condition	::=	NOT (condition) relation-condition
logical-operator	::=	AND OR
relation-condition	::=	arithmetic-expression relational-operator arithmetic-expression
relational-operator	::=	> < [NOT] = >= <=
identifier	::=	data-name [(subscript { , subscript })]
subscript	::=	integer index-name [adding-operator integer]
arithmetic-expression	::=	times-div { adding-operator times-div }
adding-operator	::=	+ -
times-div	::=	power { multiplying-operator power }
multiplying-operator	::=	* /
power	::=	[adding-operator] basis
basis	::=	identifier integer (arithmetic-expression)

Syntactic and Semantic Conventions

MicroCOBOL contains no { , }, [or] so these are all EBNF meta-symbols. The keywords in MicroCOBOL are capitalized. Keywords are reserved and hence cannot be used as data-names. Assume that a program-name, data-name and index-name, which are lexically the same, can only contain letters (only alphabetic characters), digits, and hyphens (-) with the restrictions that they must begin with a letter, cannot end with a hyphen and cannot have two consecutive hyphens. For example, give-2-Joe, tell-me and A45Asm3

are valid names, but 6gh, two-bad, and no-end- are not. integer is an unsigned integer. level-number is a 2-digit number in the range 01-09 (the leading 0 is required). It would be acceptable to recognize these as level-numbers instead of integers. Comments are indicated by a line beginning with a * and do not span multiple lines (i.e. each successive comment line begins with a *).

1. Determine the set of tokens which a lexical analyzer would need to recognize.
2. Design and implement a lexical analyzer procedure to read a source program in the above language and print the next token in the input stream. If the token detected is a valueless token, such as a keyword, then it is sufficient to print only the keyword. If it has a value, then both the token type and lexeme should be printed.
3. You will be given several MicroCOBOL programs with which to test your lexical analyzer. These will be placed on the class WWW page and will be of the form Test1.cbl, Test2.cbl, etc.

Suggestions:

1. Construct a token class to represent the token data structure, including a method to print a token.
2. Use the Java JLex tool to automatically construct a lexical analyzer in Java from a set of regular expressions specifying tokens. This can interface with your token class to return a token to the main program which then prints the token.