CS39003 Compilers Laboratory
Autumn 2025
Assignment 0
Date of posting: 21-Jul-2025

Introduction

In this assignment, you write an interpreter on some basic string operations. Strings are stored as named variables. Literal strings and strings stored in variables are concatenated to get larger strings. Your program enters a loop that keeps on reading lines of statements from the user, and storing the concatenated strings in named variables. The use enters *exit* to terminate the program.

Your program must not use any compiler-specific tools. You need to write a C/C++ program all by yourself.

In this assignment, we consider alphabetic strings only, that is, strings consisting of only the lower-case letters *a–z* and the upper-case letter *A–Z*. No other character can be a part of a string. We use the following notations.

- A ***string*** is either a literal string (not quoted) or a string stored in a variable. For example, *abcdef* and *ababababab* are literal strings. The string stored in a variable *var* is written as $*var*. For example, if the variable *mystr* stores the string *CompilersLab*, then $*mystr* evaluates to *CompilersLab*.

- A (*string*) ***argument*** is either a string (defined as above) or a string followed by the symbol caret ^ and then by a non-negative integer *n*. This stands for the *n*-fold concatenation of the string with itself. For example, *ab*^4 is *abababab* (but not *abbbb* as per our convention). Moreover, if the variable *S* stores the string *bla*, then $*S*^3 is the string *blablabla*. The 0-th power (*n* = 0) of any string is the empty string.

- An ***r-value*** (or a ***string expression***) is a concatenation of string arguments. The concatenation operator is . (dot). For example, *ab.c*^4 is the string *abcccc*. If *S* stores *Compile*, and *T* stores *Lab*, then $*S* . *rs* . $*T* is the string *CompilersLab*. An r-value may contain any number of string arguments. If an r-value uses an undefined variable, then the empty string is to be substituted for that reference after printing an appropriate error message.

- An ***assignment*** is of the form *l-value* = *r-value*. Here, an r-value is as defined above, whereas an l-value is the name of a string variable. We will follow the same naming convention for variables as in C. Anything else should be reported by printing an error message, and no assignment should be made in this case.

The user keeps on entering assignments (one in each line). The r-value is evaluated, and then that evaluated string is stored in a table against the given l-value. This table is called a ***symbol table***. If l-value is already defined, then its old value is replaced by the new value. If l-value is not defined, then a new entry in the symbol table is to be created. You should keep track of the number of symbols stored in the symbol table. There is no need to sort the table by the variable names. Do a linear search in the symbol table whenever a lookup for a variable name is to be carried out.

A pseudocode of the algorithm to be implemented is given below. Do not use any other algorithm. You do not have to detect errors other than those mentioned above (invalid l-values and undefined variables).

1. Read a *line* from the user.
2. Remove all white spaces (spaces and tabs) from *line*.
3. If *line* is *exit*, terminate the program.
4. Otherwise, line should be of the form *l-value*=*r-value*. If not, print an error message, and go to Step 1.
5. Check whether the *l-value* is a valid variable name. If not, print an error message, and go to Step 1.
6. Split the *r-value* into a list of string arguments (separated by the concatenation operator .).
7. For each string argument, check whether the exponentiation operator ^ is present. If so, convert the exponent to the repetition count *r*. Otherwise, take *r* = 1. Then, check whether the string in the string argument is a literal string (unquoted) or the reference to a variable (beginning with $). If it is a variable reference, make a symbol-table lookup to obtain the string (or the empty string if the variable is undefined). The string is now evaluated. It is then concatenated with itself *r* times to get the value of the string argument.
8. Concatenate all the evaluated string arguments from left to right. Call this string *rval*.
9. Make a symbol-table lookup for *l-value*. If that variable is already present in the symbol table, replace the old value of *l-value* by *rval*. Otherwise, create a new entry in the symbol table to store the (*l-value*, *rval*) pair. Go to Step 1.

Submit a single C/C++ source file.

Sample Output

Here, we use `line>` as the prompt of the program, and `$` as the shell prompt.

```
$ ./a.out
line> S1 = abc
S1 is set to "abc"
line> S2 = $S1 ^ 10
S2 is set to "abcabcabcabcabcabcabcabcabcabc"
line> T = $S1 . def^2 . $S2
T is set to "abcdefdefabcabcabcabcabcabcabcabcabcabc"
line> U = x . y^7 . z
U is set to "xyyyyyyyz"
line> V = $U ^ 3 . $W^2 . z^5 . $S1 . defghijklmnopqrstuvw
*** Undefined variable "W"
V is set to "xyyyyyyyzxyyyyyyyzxyyyyyyyzzzzzzabcdefghijklmnopqrstuvw"
line> exit
$
```