Assignment 1

Deadline:	Hand in by 5pm on Friday 31st March 2023
Evaluation:	10 marks – which is 10% of your final grade
Late Submission:	1 mark off per day late
Work:	This assignment is to be done **individually** – your submission may be checked for plagiarism against other assignments and against Internet repositories.
Purpose:	To reinforce ideas covered in the lectures for understanding and using imperative programming languages and language concepts.

Problem to solve:

Write a parser/interpreter for a simple, imperative string-processing language.

Requirements:

The interpreter must implement the language described below and read input from the standard input (`stdin`) and write output to the standard output (`stdout`). The interpreter should read statements from the user one at a time and perform the associated action immediately. If invalid input is provided, the interpreter should attempt to recover and try to provide a useful error message.

Strings Language:

The EBNF grammar for the strings processing language is given below, note that **`bold`** names are used for EBNF tokens and *`italics`* names are used for tokens from the lexer (defined below).

```
program    := { statement }
statement  :=  append id expression end
           |   list end
           |   exit end
           |   print expression end
           |   printlength expression end
           |   printwords expression end
           |   printwordcount expression end
           |   set id expression end
           |   reverse id end
expression :=  value { plus value }
value      := id | constant | literal
```

The (incomplete) regular expressions for the tokens in the language are given below. Please note that if you use a language regular expression library then you may need to adjust the syntax for the expressions below. E.g. the '+' character is a special character in regular expressions. You may also need to use syntax to only match characters at the start of the input.

```
append         = append
exit           = exit
list           = list
print          = print
printlength    = printlength
printwords     = printwords
printwordcount = printwordcount
reverse        = reverse
set            = set
constant       = SPACE|TAB|NEWLINE
end            = ;
plus           = +
id             = [a-zA-Z][a-zA-Z0-9]*
literal        = <for you to determine>
```

Literals:
The string literals in this language are enclosed with double-quote characters ("...") and may contain:

- letters - upper-case or lower-case
- digits
- whitespace - spaces, tabs or newlines
- punctuation - all standard punctuation marks including ';'
- special characters - special characters such as double-quotes if escaped with a backslash

Commands:
The intended behavior of each instruction is given in the following table:

Command	Parameters	Behaviour
`append`	*`id`* **`expression`**	Evaluate the expression and append it to the variable contents.
`list`		List all variables and their contents.
`exit`		Exit the interpreter
`print`	**`expression`**	Evaluate and print the expression
`printlength`	**`expression`**	Evaluate the expression and print its length
`printwords`	**`expression`**	Evaluate the expression and print the individual words
`printwordcount`	**`expression`**	Evaluate the expression and print the number of words
`set`	*`id`* **`expression`**	Set the contents of the variable to the expression
`reverse`	*`id`*	Reverse the order of the words in the contents of the variable.

Words:
Some of the commands require words to be identified within a string. For this language, a word consists of any set of letters or digits separated by whitespace (or punctuation characters). The only exceptions to this are words that contain a single-quote or hypen character. For example, *"let's"* or *"runner-up"* are considered a single word.

Notes:
Your program should read from the standard input and write to the standard output, your program should not open or close any files.

One of the first things you should do is determine what language you are going to use for your program. The assignment can be completed in most languages, but some may make it easier than others. A good regular expression engine may be helpful.

The first major part of the assignment is to write a parser. You may want to consider a **recursive descent parser** to base your code on. This type of parser looks at the next token from the lexer and from this can decide which rule to match. For example, if the statement starts with the *print* token then the parser should expect to parse an *expression* followed by an *end* token.

The other main part of the assignment is writing the interpreter that will execute instructions from the program it reads. To do this, you will need to decide what **data structures** you will use for your **symbol table** which will store all the **variables** in your program (and provide a way to access their contents). This language has only **one implicit type** (a string), so your symbol table does not need to store any information about the type of the variable.

Decisions:

There are some features that have been left (intentionally) unspecified.

- The regular expression for string literals has not been defined. How are you going to match string literals that contain punctuation?
- How is your interpreter going to respond when the user enters invalid input? Your program should try to provide some type of error handling and should not just crash. A common approach is just to read and ignore all input until it reaches the next *end* token (the ';' character).

These decisions are left up to you to make but you should document them within your assignment comments.

Sample Behaviour:

Sample input/output from the interpreter - input typed by the user is highlighted in **bold** (see the Stream Site for more examples).

```
----------------------------------------
159.341 2023 Semester 1, Assignment 1
Submitted by Rick Deckard, 20191187
----------------------------------------
set one "The cat";
set two "sat on the mat";
set sentence one + SPACE + two;
append sentence " by itself.";
print sentence;
The cat sat on the mat by itself.
printwordcount sentence;
Wordcount is: 8
printwords sentence;
Words are:
The
cat
sat
on
the
mat
by
itself
printlength sentence;
Length is: 33
list;
Identifier list (3):
one: "The cat"
two: "sat on the mat"
sentence: "The cat sat on the mat by itself."
reverse one;
print one;
cat The
exit;
```

1. Place the following comments at the top of your program code and **provide the appropriate information**:

```
Family Name, Given Name, StudentID, Assignment number, 159.341
explain what the program is doing . . .
```

2. Ensure that your program prints this information to the console. You might use code like:

```
"----------------------------------------"
"  159.341 Assignment 1 Semester 1 2023  "
"  Submitted by: Rick Deckard, 20191187  "
"----------------------------------------"
```

Hand-in: Submit **your program (as a zip file)** electronically through the form on the stream site. Marks will be allocated for correctness, fitness of purpose, sensible use of data structures/algorithms, use of language features and sensible **comments**. Good comments will help me to award you marks even if your code is not quite perfect.

If you have any questions about this assignment, please ask the lecturer.