Assignment 1

Deadline:	Hand in by 5pm on Monday 16th April 2018
Evaluation:	15 marks – which is 15% of your final grade
Late Submission:	2 Marks 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. If you adapt material from the internet you must acknowledge your source.
Purpose:	To reinforce ideas covered in the lectures for understanding and using imperative programming languages and language design concepts.

Problem to solve:
Write a String Processing Interpreter program in **ANSI C** (**not** C++) that conforms to the EBNF listed below, and which reads from `stdin`, and writes to `stdout`. Search for `gcc ansi` for more information about how to compile ANSI C.

Requirements:

```
<program> := <statement>*
<statement> := append <identifier> <expression> ;
             |  list ;
             |  exit ;
             |  print <expression> ;
             |  printlength <expression> ;
             |  printwords <expression> ;
             |  printwordcount <expression> ;
             |  set <identifier> <expression> ;
             |  reverse <identifier>;
<expression >:=  <value> { + <value> }*
<value> := <identifier> | <constant> | <literal>
<constant> := SPACE | TAB | NEWLINE
<literal := " { letter | digit | punctuation }* "
<identifier> := <letter> { <letter> | <digit> }*
<letter> := A | B | ... | Z | a | b | ... | z
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<punctuation> := . | , | : | ; | ? | ! | ...
```

You **must** follow the next two specifications in **each and every assignment for this course**

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

```
/* Family Name, Given Name, Student ID, Assignment number, 159.331 */
/* explain what the program is doing . . . */
```

2. Ensure that your `main` function uses `printf` to print this information to the console. You might use code like:

```
int main( int argc, char * argv[] ){
   printf( "----------------------------------------" );
   printf( "  159.331 Assignment 1 Semester 1 2018  " );
   printf( "  Submitted by: Rick Deckard, 20191187  " );
   printf( "----------------------------------------" );
   ...
}
```

Hand-in: Submit **your program and documentation (a zip file is acceptable)** electronically through the form on the stream site.
Marks will be allocated for: correctness, fitness of purpose, sensible use of data structures and algorithms, utility, style, use of sensible **comments and program documentation**, and general elegance. 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.

Additional Notes on the Technical Specification and Hints:
Your program can just read from `stdin` and write to `stdout`, it need not open and close any files. You might want to use `stderr` for error messages if it finds input situations it cannot handle or runs out of resources.

More Over

One of the first things you should do in your program design is decide what **data structures** you will use and how to organise them. Think about what **functions and procedures** you will need.

Sample input might be:

```
set one "The cat";
set two "sat on the mat";
set sentence one + SPACE + two;
append sentence " by itself.";
print sentence;
printwordcount sentence;
printwords sentence;
printlength sentence;
list;
reverse  one;
print  one;
exit;
```

From which output **like** the following could be expected:

```
---------------------------------------
159.331 2018 Semester 1, Assignment 1
Submitted by Rick Deckard, 20191187
---------------------------------------
The cat sat on the mat by itself.
Wordcount is: 8
Words are:
The
cat
sat
on
the
mat
by
itself.
Length is: 33
Identifier list (3):
one: "The cat"
two: "sat on the mat"
sentence: "The cat sat on the mat by itself."
cat The
```

This language has only one (implicit) type - a string. Therefore, you can identify a string and it is implicitly declared and useable from there onwards. Remember strings are really sets even if we have to implement then as arrays. Notice that we have got around the need for integers by only specifying procedures that output anything that might involve an integer such as the length of a string or the number of words in it. The language therefore does not need an integer type.

We have only specified a single operator, the binary + operator for concatenation. Think of + as a function that takes two strings and returns a third which is the concatenation of the two arguments. In your C program you might define the constants as:

```
SPACE  as  " "
TAB as "\t"
NEWLINE  as "\n"
```

How will you store strings internally? You can use any of the ANSI C strings facilities. Recall, they are implemented as char arrays with the convention of a `'\0'` termination character in the last place.
You might use the `putc/getc` and read everything character by character, which would allow you do deal with extra long lines of input. The `ispunct`, `isdigit` and `isletter`, `strcpy`, `strtok` functions may be useful to you.

You might keep a symbol table of identifiers as your program reads its input. As you encounter each identifier in the input just check to see if your symbol table already contains it, if not add it. Do an exact lexical match - so that identifiers are case sensitive in this language.

Note that the statements have a terminating semicolon. This means you can be sure when you have enough input to parse them. How will you break down expressions? What associativity convention will you use for the + operation?
Think about these specifications – when you implement them what will you do about any ambiguities?