Lab 1: String processing and getting started with lexing-parsing

1. Write a non-recursive program that prints all permutations of an input string. For instance,

```
$ ./a.out  xyz
xyz xzy yxz yzx zxy zyx                 // any order should do.
```

2. Write a program in 1.c compiling to 1.out such that the following happens:

```
$ gcc 1.c -o 1.out
$ 1.out   1.c
1                                       // prints 1
generated file 2.c                      // creates a file 2.c in the current directory
$ gcc 2.c -o 2.out
$ 2.out   2.c
2                                       // prints 2
generated file 3.c                      // creates file 3.c
$ gcc 3.c -o 3.out
$ 3.out   3.c
3
generated file 4.c
...
```

3. Write a program without using tools for lexing and parsing (such as lex and yacc) to parse variable declarations and initializations in C. Use functions from string.h such as strtok and macros from ctype.h such as isalpha.

```
Grammar:
Decl: Type  VarList ;
Type: int | char | double         // we will not support others.
VarList: Var | Var ,  VarList
Var: [_A-Za-z][_A-Za-z0-9]*
```

Print Valid or Invalid. Run it on the following input lines:

```
int a;                  // Valid
int          a;         // Valid
a int;                  // Invalid
integer a  ;            // Invalid
int double a;           // Invalid
int a                   // Invalid
int int;                // Invalid
int ;                   // Invalid
int 10;                 // Invalid
int 10a;                // Invalid
int a, b , a, b;   // Invalid in C, but you may also mark it syntactically Valid if you wish.
int CS16B000, cs16b000;     // Valid
int a, ab, abc, abcd;       // Valid
```

4. Learn basic lex and yacc and redo Problem 3 using the tools. Compare.