# CSEN 4111 - Odd Sem 2020 End Sem Lab Exam

*Answer only the question allotted to you*

1. Write a translator to separate out simultaneous variable declaration cum assignment statements.

   Example:

Table 1:

| Input | Output |
|---|---|
| int a; | int a; |
| int b=a; | int b; |
| | b=a; |
| c=b; | error |

2. Write a translator for the following language:

   `int arr;` declare the storage "arr" for conducting the operation
   `OP DIR x` ; Use OP and move DIR direction, meaning of x depends on on the operation OP
   OP codes can be:
   *W means write to array,*
   *R read means read from array,*
   *P means print the array,*
   *M means move over the array*
   *Each operation takes place on the current head*

```
DIR can be L, means move left 1 pos (default)
R means move right by 1 pos (default)
S means stay at the cuyrrent position

'x' in case of Opcode 'W' indicates the value to be written to the storage
'x' has no meaning in case of Opcodes 'R' and 'P'
'x' in case of 'M' opcode means move by those many positions left or right depending
on the DIR code
```

   Example:

```
int arr;
W R 2           =>  arr[0] = '2' , current pointer is set to '1'
M L 1           => set current pointer to '0'
R R             => print arr[0]
W R 3           => arr[1]='3', current pointer set to '2'
P               => print the whole configuration of arr
```

   Your translator should print the current configuration of the array after each statement execution. You may make any assumption to make sure your program works correctly under various boundary conditions.

3. Write a translator for handling autoincrement operators in a language that lacks these supports:

   Example:

Table 2:

| Input | Output |
|---|---|
| ++b; | b=b + 1; |
| b++; | b=b + 1; |
| --b; | b=b - 1; |
| b--; | b=b - 1; |

4. Write a translator for simplifying the chain of assignment operators in the order shown in the example:
   Example:

Table 3:

| Input | Output |
|---|---|
| a=b=c=d; | c=d;<br>b=c;<br>a=b; |

5. Write a translator for a langauge that simplifies the chain of string addition operations as shown in the example:
   Example: (assume all the variables are declared char* i.e. String )

Table 4:

| Input | Output |
|---|---|
| a=b+c+d; | strcpy(a,b);<br>strcat(a,c);<br>strcat(a,d); |

6. Write a translator for an assembly langauge to rearrange code to exploit **Delayed Branching** as shown below:

   There are two types of instructions in the assembly language.
   One type is of the form `OPn mem,Rm`
   `Or, OPn Rm,mem`, where n,m are *int*.
   *mem* refers to memory variables, and can be a combination of alphabets only.
   *Rn* refers to any one of the available registers.
   *OPn* referes to any of the mathematical operators with two opernds, one of them is a memory variable and the other one is a register.

   The other type of instruction is of the form `JNZ Rn,Lm`, where *Ln* refers to a `Label` to jump to.
   The idea of Delayed Branching is to push the JNZ instruction as far back as possible, and bring OP instructions as far front as possible (maintaining their internal sequence). Of course you cannot bring an OP instruction before a JNZ instruction if they are using the same register.

   Example:

   Table 5:

| Input | Output |
|---|---|
| L1: OP1 a,R1 | L1:OP1 a,R1 |
| OP2 b,R2 | OP2 b,R2 |
| JNZ R2,L1 | OP3 c,R3 |
| OP3 c,R3 | JNZ R2,L1 |

   But you cannot do the above transformation if OP3 was using R2.
   You may make any assumption about your translator but make sure you write these cclearly.