Question 1

Note: You should only use the predefined/provided functions. You are to assume that functions such as printf and its equivalent are not available. The outputs of your answers to all parts of this question should end at the last digit, and not on a newline.

Given the code for function

```
printlnX(int x)
{
        for(int i = 0; i < x; i++)
                printNumStr (i, ", ");
}
```

Such that the execution of *printlnX(3)* outputs

```
0, 1, 2,
```

(a) This function prints a trailing comma, making it visually unpleasant. Modify the function above to remove the last ',' in its output.

(b) Define the code for function *printMatrixX(int x)*. For instance, *printMatrixX(3)* should output:

```
0, 1, 2
0, 1, 2
0, 1, 2
```

And *printMatrixX(4)* should output:

```
0, 1, 2, 3
0, 1, 2, 3
0, 1, 2, 3
0, 1, 2, 3
```

(c) Define the code for function *printPyramid(int x)*.

For instance, *printPyramid(3)* should output:

```
0
0, 1
0, 1, 2
0, 1
0
```

And *printPyramid(4)* should output:

```
0
0, 1
0, 1, 2
0, 1, 2, 3
0, 1, 2
0, 1
0
```

(d) In a particular program, a list of numbers can be expressed as a sequence (e.g. [2, 2, 1, 4, ...]).

You are given two additional functions:

(i) *item(list a, int n)* returns you the n^{th}+1 element in the list a. e.g., *item([1,2,3,4,5], 2)* returns 3

(ii) *length(list a)* returns the length of the list. e.g., *length([1,2,3,4,5])* returns 5

Define the code for function *printRidge(list a)* based on the given functions above.

For instance, *printRidge([2, 1, 4])* should output

```
0
0, 1
0
0
0
0, 1
0, 1, 2
0, 1, 2, 3
0, 1, 2
0, 1
0
```

Another example is *printRidge([3, 2])* should output

```
0
0, 1
0, 1, 2
0, 1
0
0
0, 1
0
```

Question 2

Note: Syntax used in this question belongs to a class programming languages known as functional programming.

Assume that your computer only has two built in unary operators, ++ and --, when applied to a variable, *a*, the application of *a++* gives the value of *a+1* and *a--* gives value of *a-1*. For instance, when *a* stores the value 2, *a++* evaluates to 3 and *a--* evaluates to 1.

We define the following *add* function that takes two numbers and return their sum.

```
add 0 y = y
add x 0 = x
add x y
        | (x < y) = add y x
        | otherwise = add (x--) (y++)
```

(a) This function doesn't seem to terminate for most values, describe and fix the error.

Hint: describe what happen when add 5 3 is executed; show each path or call that is executed.

Your computer is upgraded with three more unary operators:

Operator	Description	Example
a»	Returns a×2	When a is 5, a» evaluates to 10
a«	Returns $\lfloor a \div 2 \rfloor$	When a is 5, a« evaluates to 2
??	??	??

The documentation for the last operator was lost. You now get to design the functionality of this last operator.

(b) You know that these three operators allow you to define a better and faster *add* function.

(i) Design this missing operator.
Hint: There are more than one possible solution, you only need provide one. You are not allowed to define an operator that magically adds two numbers

(ii) Using these three operators, define a better *add* function. Describe why your function is better.

(iii) Show what happen when your *add* function works on add 17 13

You are given another operator:

Operator	Description	Example
-a	Negates the sign of an numeric variable	When a is 5, -a evaluates to -5

(c) Using the - operator, modify your *add* function to handle negative values.

Question 3

EIGHT-V is a processor with 8 32-bits wide general-purpose registers named V0 to V7, and a special register: PC, which represents the current program counter. Each instruction is 32-bits (4 bytes) wide.

It has the following symbolic instructions:

Instruction	Description
`a Va Vb -> Vout`	Vout = Va + Vb (unsigned)
`a Va #n -> Vout`	Vout = Va + n (Where n is a literal)
`m Vb -> Va`	Va = Vb
`m #n -> Va`	Va = n (Where n is a literal)
`ld d(Vb) -> Va`	Va = memory[Vb]
`st Vb -> ad(Va)`	memory[Va] = Vb
`cp Va Vb`	Compares if Va == Vb
`j <address>`	Unconditionally set PC = <address>
`eq.j <address>`	PC = <address> if last comparison is equal, PC = PC+4 otherwise
`stop`	End the execution of the program

Execution for this processor begins at PC=0, with all registers set to 0, and all memory addresses (except for the loaded program) initialized to 0.

(a) The following program is executed:

```
start:
        m #10 -> V1
        m #100 -> V2
        m #110 -> V3

loop:
        cp V2 V3
        eq.j end
        st V1 -> ad(V2)
        a V2 #1 -> V2
        j loop
end:
        stop
```

(i) What is the value at memory location 101?

(ii) What is the value at memory location 110?

(iii) Modify the program to set the value of each memory location to itself for memory locations between 100 to 200. (i.e. the value of memory location 100 is 100, the value at 101 is 101 and so on)

(b) The following program computes $4 \times$ V1 and places the result in V3

```
m #0 -> V3
a V1 V3 -> V3
a V1 V3 -> V3
a V1 V3 -> V3
a V1 V3 -> V3
stop
```

(i) Modify the program to give instead $4 \times \text{V1} + 10$

(ii) Modify the program to place the result of V1 $\times$ V2 in V3, for any value of V1, V2.

(iii) Describe what happens when V1 = 1073741824, and V2 = 4, with the program you have written in ii