1. (10 points) Write code for a state diagram to recognize the floating-point literals, integer literals, string literals and variable names of any programming language that was developed after 1990.

https://github.com/parkhwyy/gsu-plc/blob/master/Final/final_q1.py

2. (10 points) Write an CFG, EBNF or BNF for Java; the tool you create only need to cover the following expressions and show the proper order of operations:

   –boolean expressions

   –assignment statements

   –mathematical expressions

<decls> -> int <idlist> ;
<idlist> -> id { , id }
<stmts> -> <stmt> [<stmts>]
<cmpdstmt> -> '{' <stmts> '}'
<stmt> -> <assign> | <cond> | <loop>
<assign> -> id = <expr> ;
<cond> -> if '(' <rexp> ')' <cmpdstmt> [else <cmpdstmt>]
<loop> -> while '(' <rexp> ')' <cmpdstmt>
<rexp> -> <expr> (< | > | <= | >= | == | !=) <expr>
<expr> -> <term> [(+ | -) <expr>]
<term> -> <factor> [(* | / | %) <term>]
<factor> -> int_lit | id | '(' <expr> ')'

3. (10 points) Write recursive descent parser routines for the rules created above.

https://github.com/parkhwyy/gsu-plc/tree/master/Final/final_q3

4. (10 points) Explain the four criteria for proving the correctness of a logical pretest loop construct of the form "while B do S end". And prove the correctness of the following:

```
a = 1;
b = 1;
while( b <= n ){
   a = a * x;
   b = b + 1;
}
{ a = x ^ n }
```

P: precondition, Q: postcondition, I: loop invariant,

while B do S end

1) P => I

2) { I and B } S { I }

3) (I and !B) => Q

4) The loop terminates.

1) No preconditions

2) { I and B } S { I }

{ n >= 0 } && { b <= n } // b = 1

a = a * x => S1

b = b + 1 => S2

{ n >= 0 }

{ n >= 0 } && { b <= n } // b = 1

{ n > 0 }

a = a * x => S1

b = b + 1 => S2

{ n >= 0 }

{ n > 0 } => { n >= 0 } TRUE

3) (I and !B) => Q

{ n >= 0 } && { b > n }

=> { a = x ^ n } // a = 1

{ 0 <= n < b } // b = 1

{ 0 <= n < b }

{ 0 = n } => { a = x ^ n } = { a = x ^ 0 }

= { a = 1 } => { 1 = 1 }

4) The loop terminates.

n >= 0

while (b < = n) { => B

a = a * x => S1

B = b + 1 => S2

}

b = 1

n > 0

while (b < = n) { => B

a = a * x => S1

b = b + 1 => S2

}

0 < n

b <= n

b < b + 1 < . . . < b + x

for some x where b + x = n + 1

b = 1

n > 0

while (b < = n) { => B

a = a * x => S1

b = b + 1 => S2

}

Never executes

5. (5 points) In a letter to the editor of CACM, Rubin (1987) uses the following code segment as evidence that the readability of some code with gotos is better than the equivalent code without gotos. This code finds the first row of an n by n integer matrix named x that has nothing but zero values.

```
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++)
        if (x[i][j] != 0)
            goto reject;
    println ('First all-zero row is:', i);
    break;
    reject:
}
```

Rewrite this code without gotos in Java. Compare the readability of your code to that of the example code.

https://github.com/parkhwyy/gsu-plc/blob/master/Final/final_q5.java

```
boolean found = false;

for (int i = 1; i <= n; i++) {
        int counter = 0;
        for (int j = 1; j <= n; j++) {
                if (x[i][j] == 0)
                        counter++;
        }
        if (counter == n && found == false) {
                System.out.println("First all-zero row is: " + i);
                found = true;
        }
}
```

Using goto makes the code more readable and easier to follow since it provides the shorter and more straightforward way to express the logic. The code without goto is a bit longer and might be more difficult to understand.

6. (10 points) Consider the following programming problem: The values of three integer variables—first, second, and third—must be placed in the three variables max, mid, and min, with the obvious meanings, without using arrays or user-defined or predefined subprograms. Write two solutions to this problem, one that uses nested selections and one that does not. Compare the complexity and expected reliability of the two.

https://github.com/parkhwyy/gsu-plc/tree/master/Final/final_q6

The program with nested if conditions would have a control flow that is difficult to follow. It is almost impossible to tell which code will run, or when, resulting in more complexity. Both readability and writability influence reliability. A program that does not follow natural ways to express the required algorithms will necessarily use unnatural approaches. Unnatural approaches are less likely to be correct for all possible situations. The easier a program is to write, the more likely it is to be correct. Readability affects reliability in both the writing and maintenance phases of the life cycle. Programs that are difficult to read are difficult both to write and to modify. For this reason, the one without the nested if conditions should be considered more reliable.

7. (5 points) Compare the tombstone and lock-and-key methods of avoiding dangling pointers and memory leakage, from the points of view of safety and implementation cost.

A tombstone method is used to catch the memory access erros to objects through stacks and heap. Tombstone is allocated whenever an object is allocated in a heap or in a stack. Here the pointer consists of the address of the tombstone, where tombstone consists of the address of the object. When the object is claimed back, then the modification of tombstone is done such that tombstone contains a value (zero) which is invalid address. Tombstone are created as static objects in order to avoid important cases in the created code. When the program calls the deallocation operation, heap objects can easily nullify tombstone. To validate a tombstone, it is better to link all stack objects together in a list which are sorted by the address of the stack frame in which the object lies.
Hence, tombstones can be expensive both in time and in space.
Possible Overheads :
(1) Allocation : When allocating heap objects or using a pointer to operator to make tombstones
(2) Valid Check : It checks for validity on every access.
(3) Double indirection : It is time-consuming since the location of an object present in the heap changes quickly.
(4) Size of Tombstone : A long-running program that creates and claim back objects continuously will ultimately run out of space for tombstones. Although the size of tombstone is less than the object to which it refers but its creation and claiming back continuously will increase the space overhead.

Lock-and-key is an alternative to a tombstone method. A probabilistic protection from memory access errors is provided as it works for objects in the heap. Lock-and-key is simple as it provides an address and a key to every pointer in a tuple. Each object in the heap has a lock in the beginning. When the key in the pointer is same as that of lock in the object , the pointer to an object in the heap is considered valid. A new key value is created whenever a new heap object is allocated at runtime. When an object is claimed back, its lock is changed to some arbitrary value so that the keys in any remaining pointers does not match. If the block is continuously reused for another purpose, then the location that used to contain the lock will be restored to its former value .
Possible Overheads:
(1) Extra space : Storage is increased as it adds an extra word of storage to every pointer and to every block in the heap.
(2) Copy : As it copies one pointer into another, the cost increases. It compares locks and keys on each access, which adds an extra cost.

In terms of implementation cost, it is difficult to say which is more expensive. A tombstone method may result in two cache misses (one for the tombstone and one for the object), while lock-and-key copies pointers and makes comparisons on each access, increasing the cost. In terms of safety, lock-and-key is more secure as it provides the probabilistic protection.

8. (10 points) Consider the following C program segment.

```
j = -3;
for (i = 0; i < 3; i++) {
   switch (j + 2) {
     case 3:
     case 2: j--; break;
     case 0: j += 2; break;
         default: j = 0;
   }
   if (j > 0) break;
   j = 3 - i
}
```

a)Rewrite it using no gotos or breaks.

b)Rewrite the C program segment using if and goto statements in C

c) Rewrite the C program segment in Java without using a switch statement

d)Write and explain the operational semantics C program segment

https://github.com/parkhwyy/gsu-plc/tree/master/Final/final_q8

```
j = -3;
for (i = 0; i < 3; i++) {
        evaluate (i = 0)
        loop:
        control = evaluate (i < 3)
        if control == 0 goto out  // If i < 3 is false, exit the for loop.

        switch (j + 2) {
                evaluate (j + 2)
                case 3: j--; goto if  // If (j+2) == 3, j-- and exit the switch.
                case 2: j--; goto if  // If (j+2) == 2, j-- and exit the switch.
                case 0: j += 2; goto if  // If (j+2) == 0, j += 2 and exit the switch.
                default: j = 0  // If (j+2) is any other values, j = 0 and exit the switch.
        }

        if:
        if (j > 0) {
                cond = evaluate (j > 0)
                if cond == 1 goto out  // If j > 0 is true, exit the for loop.
        } else {
                j = 3 - i
        }

        evaluate (i++)
        goto loop
}
out:
return 0
```

9. (30 points) Analyze a language introduced in the last 15 years, for its:

   readability, writability, reliability

   In this analysis you must also address the following, and the problems that are introduced by making the choices that language did:

   Keywords

   Data type

   Control Structures

   Expressions

   - unary, binary, trinary, combinations

   - assignment

   - logic

   - order or operations

   Compare the selected language to Java .

   For 10points extra credit analyze how the language you chose handles syntax and semantics

A separate PDF file has been attached to answer this question.

(Swift Analysis.pdf)