```python
#------------------------------ Domain Class-----------------------------------
class Domain():
    def __init__(self, name, start, end):
        self.name = name
        self.values = range(start,end+1)
        self.vars = []


    def addVariable(self,variable):
        self.vars.append(variable)

    def getValues(self):
        return self.values

    def getVars(self):
        return vars

    def strSelectVariables(self):
        print("vars"+str(self.vars))
        res = ""
        for v in self.vars:
            res += "?" + v + " "
        return res

    def __str__(self):
        res = ":" + self.name + " rdf:type :Domain ;\n" + "\t:values "
        for val in self.values[:-1]:
            res += str(val) + ", "
        res += str(self.values[-1]) + " ;\n\t:variables"
        for var in self.vars[:-1]:
            res += " :" + str(var) + ","
        res += " :" + self.vars[-1] + ".\n\n"
        return res



#--------------------------- Constraint Class----------------------------------
class Constraint:
    def __init__(self):
        self.typeCons = ""
        self.vars = []
        self.values = []
        self.first = ""
        self.second = ""
        self.third = ""

    def addVar(self, var):
        self.vars.append(var)

    def addValue(self, value):
        self.values.append(value)

    def setTypeCons(self,typeCons):
        self.typeCons = typeCons
        if self.typeCons == "Reject:\n":
            self.first = " != "
            self.second = " || "
            self.third = " && "
        elif self.typeCons == "Accept:\n":
            self.first = " = "
            self.second = " && "
            self.third = " || "

    def __str__(self):
        print(self.vars)
        print(self.values)
        res = "\t\t( "
        for i in range(len(self.values)):
            res += "("
            for j in range(len(self.vars)):
                res += "?" + str(self.vars[j]) +  self.first +  str(self.values[i][j])
                if j == len(self.vars) - 1:
                    res += ")"
                else:
                    res += self.second

            if i != len(self.values) - 1:
                res += self.third
        res += " )\n"
        return res




#---------------------------- Script itself ----------------------------------
class MainRun:
    def __init__(self):
        self.inFileName = ""
        self.outFileName = ""
        self.domains = {}
        self.fileOutRDF = None
        self.fileOutSPAQRL = None

    def writeDomains(self):
        for d in self.domains.keys():
            self.fileOutRDF.write(str(self.domains[d]))

    def parseConstraints(self, file, nConst):
        nConstParsed = 0
        for line in file:
            constraint = Constraint()
            if("Vars:" in line):
                if nConstParsed < nConst:
                    nVars = int(file.readline())
                    for x in range(nVars):
                        var = file.readline().rstrip('\n')
                        constraint.addVar(var)
                    typeCons = file.readline()
                    constraint.setTypeCons(typeCons)
                    nValues = int(file.readline())
                    for x in range(nValues):
                        lineValue = file.readline().rstrip('\n')
                        constraint.addValue(lineValue.split())
                    if nConstParsed + 1 < nConst:
                        self.fileOutSPAQRL.write(str(constraint) + "\t\t&& \n")
                    else:
                        self.fileOutSPAQRL.write(str(constraint))
                    nConstParsed += 1

    def writeSelect(self):
        self.fileOutSPAQRL.write("SELECT ")
        for _ , value in self.domains.items():
            self.fileOutSPAQRL.write(value.strSelectVariables())
        self.fileOutSPAQRL.write("\n")

    def writeWhere(self):
        self.fileOutSPAQRL.write("WHERE {\n")
        for key , value in self.domains.items():
            listVar = value.vars
            for v in listVar:
                self.fileOutSPAQRL.write("\t:" + key + " :values ?" + v + ".\n")
        self.fileOutSPAQRL.write("\tFILTER (\n")

    def run(self):
        self.inFileName = input("Enter F2CSP file name:")
        self.outFileName = input("Enter output file name:")
        self.fileOutRDF = open(self.outFileName + ".ttl","w+")
        self.fileOutRDF.write("@prefix : <http://www.w3.org> .\n")
        self.fileOutRDF.write("@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .\n\n")
        self.fileOutSPAQRL = open(self.outFileName + ".rq","w+")
        self.fileOutSPAQRL.write("PREFIX : <http://www.w3.org>\n")
        self.fileOutSPAQRL.write("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n")
        fileIn = open(self.inFileName, "r")
        for line in fileIn:
            if("Domains:" in line):
                nDomains = int(fileIn.readline())
                for _ in range(nDomains):
                    currD = fileIn.readline()
                    d = currD.split()
                    self.domains[d[0]] = Domain(d[0], int(d[1][0]),int(d[1][-1]))
            if("Variables:" in line):
                nVars = int(fileIn.readline())
                for _ in range(nVars):
                    currV = fileIn.readline()
                    v = currV.split()
                    self.domains[v[1]].addVariable(v[0])
                self.writeDomains()
            if("Constraints:" in line):
                self.writeSelect()
                self.writeWhere()
                self.parseConstraints(fileIn,int(fileIn.readline()))
                self.fileOutSPAQRL.write("\t)\n")
                self.fileOutSPAQRL.write("}")
        fileIn.close()
        self.fileOutRDF.close()
        self.fileOutSPAQRL.close()
        print("SCRIPT END")



scriptRun = MainRun()
scriptRun.run()
```