```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.lower())

    def getValues(self):
        return self.values

    def getVars(self):
        return vars

    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,varDom):
        self.typeCons = ""
        self.vars = []
        self.values = []
        self.varDom = varDom

    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 = ""
        for i in range(len(self.values)):
            if self.typeCons == "Reject:\n":
                res += " ObjectComplementOf("
            if len(self.vars) > 1:
                res += " ObjectIntersectionOf("
            for j in range(len(self.vars)):
                res += " ObjectHasValue("
                res += ":" + self.vars[j].lower()
                res += " "
                res += ":" + self.varDom[self.vars[j]].lower() + "val" +
                str(self.values[i][j]).lower() + ")"
            if len(self.vars) > 1:
                res += ")"
            if self.typeCons == "Reject:\n":
                res += ")"

        return res




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

    def parseConstraints(self, file, nConst):
        if nConst > 1:
            self.fileOutOWL.write(" ObjectIntersectionOf(")
        nConstParsed = 0
        for line in file:
            constraint = Constraint(self.varDom)
            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())
                    self.fileOutOWL.write(str(constraint))
                    nConstParsed += 1
        if nConst > 1:
            self.fileOutOWL.write(")")


    def writeHashTagSeparator(self,title):
        self.fileOutOWL.write("############################\n")
        self.fileOutOWL.write("#    " + title + "\n")
        self.fileOutOWL.write("############################\n\n")


    def writeObjectProperties(self):
        self.writeHashTagSeparator("Object Properties")
        for _ , d in self.domains.items():
            for v in d.vars:
                self.fileOutOWL.write("# Object Property: :" + v + " (:" + v + ")\n\n")
                self.fileOutOWL.write("FunctionalObjectProperty(:" + v + ")\n")
                self.fileOutOWL.write("ObjectPropertyDomain(:" + v +" :Var)\n")
                self.fileOutOWL.write("ObjectPropertyRange(:" + v + " :" + d.name + ")\n\n")

    def writeClasses(self,fileIn):
        self.writeHashTagSeparator("Classes")
        for _ , d in self.domains.items():
            self.fileOutOWL.write("# Class: :" + d.name + " (:" + d.name + ")\n\n")
            self.fileOutOWL.write("EquivalentClasses(:" + d.name + " ObjectOneOf(")
            for val in d.values:
                self.fileOutOWL.write(":" + d.name.lower() + "val" + str(val) + " ")
            self.fileOutOWL.write("))\n\n")
        self.fileOutOWL.write("# Class: :Fml (:Fml)\n\n")
        self.writeFml(fileIn)
        self.fileOutOWL.write("# Class: :Var (:Var)\n\n")
        for _ , d in self.domains.items():
            for v in d.vars:
                self.fileOutOWL.write("EquivalentClasses(:Var ObjectExactCardinality(1 :" + v + " :" + d.name + "))\n")
        self.fileOutOWL.write("\n\n")

    def writeNamesIndividuals(self):
        self.writeHashTagSeparator("Named Individuals")
        for _ , d in self.domains.items():
            for val in d.values:
                self.fileOutOWL.write("# Individual: :" + d.name.lower() + "val" + str(val) + " (:" + d.name.lower() + "val" + str(val) + ")\n\n")
                self.fileOutOWL.write("ClassAssertion(:" + d.name + " :" + d.name.lower() + "val" + str(val) + ")\n\n")
        self.fileOutOWL.write("# Individual: :fml (:fml)\n\n")
        self.fileOutOWL.write("ClassAssertion(:Fml :fml)\n")
        self.fileOutOWL.write("SameIndividual(:fml :map)\n\n")
        self.fileOutOWL.write("# Individual: :map (:map)\n\n")
        self.fileOutOWL.write("ClassAssertion(:Var :map)\n")
        self.fileOutOWL.write("\n\n")
        for _ , d in self.domains.items():
            self.fileOutOWL.write("DifferentIndividuals(")
            for val in d.values:
                self.fileOutOWL.write(":"  + d.name.lower() + "val" + str(val) +  " ")
            self.fileOutOWL.write(")\n")

    def writeFml(self,fileIn):
        self.fileOutOWL.write("EquivalentClasses(:Fml")
        n = int(fileIn.readline())
        self.parseConstraints(fileIn,n)
        self.fileOutOWL.write(")\n\n")


    def run(self):
        self.inFileName = input("Enter F2CSP file name:")
        self.outFileName = input("Enter output file name:")
        self.fileOutOWL = open(self.outFileName + ".owl","w+")
        self.fileOutOWL.write("Prefix(:=<http://www.semanticweb.org/grupo21/ontologies/2019/4/grupo21#>)\n")
        self.fileOutOWL.write("Prefix(owl:=<http://www.w3.org/2002/07/owl#>)\n")
        self.fileOutOWL.write("Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)\n")
        self.fileOutOWL.write("Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)\n")
        self.fileOutOWL.write("Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)\n")
        self.fileOutOWL.write("Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)\n")
        self.fileOutOWL.write("\n\nOntology(<http://www.semanticweb.org/grupo21/ontologies/2019/4/" + self.outFileName + ">\n\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]))
                for d in self.domains.keys():
                    self.fileOutOWL.write("Declaration(Class(:" + d + "))\n")
                self.fileOutOWL.write("Declaration(Class(:Fml))\n")
                self.fileOutOWL.write("Declaration(Class(:Var))\n")

            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.varDom[v[0]] = v[1]
                for _ , d in self.domains.items():
                    for v in d.vars:
                        self.fileOutOWL.write("Declaration(ObjectProperty(:" + v + "))\n")
                for _ , d in self.domains.items():
                    for val in d.values:
                        self.fileOutOWL.write("Declaration(NamedIndividual(:" +
                        d.name.lower() + "val" + str(val) + "))\n")
                self.fileOutOWL.write("Declaration(NamedIndividual(:fml))\n")
                self.fileOutOWL.write("Declaration(NamedIndividual(:map))\n")
                self.writeObjectProperties()
                self.fileOutOWL.write("\n")

            if("Constraints:" in line):
                self.writeClasses(fileIn)
                self.writeNamesIndividuals()
        self.fileOutOWL.write(")")
        fileIn.close()
        self.fileOutOWL.close()
        print("SCRIPT END")



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