프로그래밍 실습 1

```
from __future__ import print_function
from sys import stdin, stdout, exit
from math import ceil
from collections import OrderedDict
import re

def printf(str, *args):
	print(str % args, end='')

# 여러 출력 대상(파일, 표준출력) 같은 내용을 출력하기 위한 클래스입니다.
class Writer:
	def __init__(self, *writers):
		self.writers = writers

	def write(self, text):

		for w in self.writers:
			l = w.write(text)
		return l

# 가상으로 구현된 스택의 정보처리 의 한 단위를 의미하는 Node입니다.
class Node:
	def __init__(self, value = 0, headarrive = 0, modify="", next=None):
		self.Value = value
		# 합성명제 또는 피연산자의 진리값을 담는 변수입니다.
		self.HeadArrive = headarrive
		# Stack의 head가 비어있는지 비어있지 않은지를 판정하는 변수입니다.
		self.Modify = modify
		# 합성명제 식 또는 피연산자, 연산자들이 들어있는 문자열 공간입니다.
		self.Next = next
		# 다음 노드를 가리키는 구조체 포인터입니다.

	def Push(head, value, modify = ""):
		Temp = head
		if head.HeadArrive == 0:
			head.Value = value
			head.Next = None # 최상단 노드이기때문에 Next는 Null(Python에서는 None)입니다.
			head.HeadArrive = 1
			head.Modify = modify
			return head # 헤드값을 리턴합니다.
		else:
			while Temp.Next != None: # 스택의 최상단까지 찾아갑니다.
				Temp = Temp.Next
			NewNode = Node(value, 0, modify) # 새로운 노드를 할당합니다.
			Temp.Next = NewNode
			return NewNode

	# Top노드를 팝시키고 Top Node 전 주소값을 리턴합니다.
	# C 버전 코드와 달리 copyModify 인자를 쓰지 않습니다.
	def Pop(head):
		Temp = head
		if head.Next == None:
			if head.HeadArrive == 0:
				printf("잘못된 Pop연산.₩n")
```

```
                        return None
                head.HeadArrive = 0
                return [head, head] # 최상단값인 헤드를 리턴합니다.
        else:
                while Temp.Next.Next != None:
                        Temp = Temp.Next
                Temp2 = Temp # Top노드의 전노드주소를 저장합니다.
                Temp = Temp.Next # 제거될 Top노드의 주소값을 저장합니다.
                Temp2.Next = None
                return [Temp, Temp2] # 제거될 노드의 주소와 Top노드 전노드의 주소를 리턴합니다.

# 이함수는 명제에서 중복되지 않는 명제변수를 찾아내는 함수임
# 중복되지 않는 명제변수의 개수를 리턴하는 C 버전 코드와 달리
# 명제변수의 문자열 EleAry를 리턴값으로 반환함
def checkElementary(Input):
        EleAry = ''.join(list(OrderedDict.fromkeys(list(re.sub('[&₩|>!]','',Input)))))
        return EleAry

# 문자열을 스택에서 꺼낼 때 반대순서로 나오기 때문에 그걸 출력할때는 거꾸로 출력해야합니다.
# 문자열 인자 cAry에 거꾸로 바꾸어 기록하는 C 버전 코드와 달리, 그 결과를 리턴하는 함수
def cSwap(cAry):
        return cAry[::-1]

# Parsing Table은 기본적으로 & | > ! 순으로 되어있습니다.
def ParsingCheck(Stack, Input):
        # Stack이 우선순위가 높다 = 1,  Input이 우선순위가 높다 = 0
        ParsingTable = [
                                                [1,1,1,0],
                                                [1,1,1,0],
                                                [1,1,1,0],
                                                [1,1,1,0]
                                        ]
        i = {'&':0, '|':1, '>':2, '!':3}[Stack] # Stack에 저장된 연산자의 Index값을 찾습니다.
        j = {'&':0, '|':1, '>':2, '!':3}[Input]
        return ParsingTable[i][j] # 각각의 Index에 해당하는 우선순위값을 ParsingTable에서 꺼내서 리턴합니다.

# 피연산자들과 연산자를 받아서 그에 해당하는 논리 연산을 마치고 리턴시켜주는 함수
def MainOperator(value_1, op, value_2):
        if op == '!':
                return (not value_1) # 만약 논리값이 0 ( false ) 일때 1 ( True ) 를 리턴합니다.
        elif op == '|':
                return (value_1 or value_2) # 두 연산자의 논리값의 Disjunction연산을 한후 1 ( True ) or 0 (
False ) 값을 리턴합니다.
        elif op == '&':
                return (value_1 and value_2) # 두 연산자의 논리값의 Conjunction연산을 한후 1 ( True ) or 0 (
False ) 값을 리턴합니다.
        elif op == '>':
                return ( (not value_1) or value_2) # 두 연산자의 논리값의 Implication연산을 한후 1 ( True ) or
0 ( False ) 값을 리턴합니다.
        else:
                printf("잘못된 연산자.₩n")
                return None

# 거듭제곱 구하는 함수 입니다.  명제변수에 따라서 출력 결과가 2^n개 이기때문에 필요.
def LowPower(under, upper):
        return under ** upper

# 명제변수의 개수와 그 크기에 해당하는 배열을 주면 Value Table만들어주는 함수입니다.
```

```
# 명제변수가 3개면 3*2^3 배열을 입력받고 그값에 F , T 를 의미하는 0,1을 세팅해주는 함수.
# C 언어 코드와 달리 명제 변수의 Value Table을 리턴합니다.
def valueTableCreate(valueNum):
	Ary = [[(1 if (j & int(LowPower(2,(valueNum-1)-i)) != 0) else 0) for j in range(0, LowPower(2, valueNum))] for i in range(0, valueNum)]
	return Ary

# 이건 명제변수와 대응되는 값을 리턴해주는 함수입니다.
def BackValue(Ary, EleAry, Check, InputNum):
	i = EleAry.find(Check) # 만약 찾고자 하는 피연산자의 Index값을 찾습니다.
	return Ary[i][InputNum] # 피연산자의 Index값에 해당하는 진리값을 리턴해줍니다.

# 진리표 T,F찍을때 명제 길이재는 함수이며 길이를 리턴합니다.
def HowManyLength(Input):
	return len(Input)

# 스택의 각 값들을 초기화 시켜줍니다.
Operator = Node()
Operand = Node()
tail_1 = Operator
tail_2 = Operand

fp = open("Truth Table.txt", 'w')
w = Writer(stdout, fp)

printf("진리표를 만듭니다. 합성명제를 입력해 주세요.₩n( 부정은 ! , 논리곱은 & , 논리합은 | , 논리함축은 > 로 표기해주세요. 괄호도 사용하시면 안됩니다.) ₩n₩n명제 : ")
InModify = stdin.readline().strip('₩n') # 명제를 입력받습니다.
EleAry = checkElementary(InModify) # 중복되지 않은 명제 변수를 문자열로 저장합니다.
valueNum = len(EleAry) # 명제변수의 개수를 세어 valueNum에 저장합니다.
valueAry = valueTableCreate(valueNum) # 명제변수의 진리값 Table을 valueAry에 생성합니다.

printf("₩n₩n")
w.write("- Truth Table -₩n₩n")

# 명제변수의 개수에 맞춰서 2 ~ 2^n회 반복합니다.
modifyPrint = 0 # 첫번째줄에는 합성명제들을 출력하기위해 사용할 변수입니다.
WholeLineLength = 0 # 총 한줄이 몇개의 문자들로 이루어져있는지 길이를 저장할 변수입니다.

j = 0
while j < LowPower(2, valueNum):
	w.write("| ")
	WholeLineLength += 2

	for k in range(0, valueNum):
		if j == 0 and modifyPrint == 0: # 명제변수의 값들을 출력합니다.
			w.write("%s " % EleAry[k])
			WholeLineLength += 2
		else:
			w.write("%s " %('T' if valueAry[k][j] else 'F') )
	w.write("|")
	WholeLineLength += 1

	for i in range(0, len(InModify)): # 명제가 끝날때까지 반복합니다.
		test = {'!':1, '&':2, '|':2, '>':2}.get(InModify[i], 0) # 명제의 원소들을 검사합니다.
		if test == 0: # C 언어 코드에서 switch 문의 default에 해당
			EleValue = BackValue(valueAry,EleAry,InModify[i],j)
			tail_2 = Node.Push(Operand,EleValue,InModify[i])
```

```
        elif test > 0:
                while Operator.HeadArrive == 1 and ParsingCheck(tail_1.Modify[0],InModify[i]):
                        TempModify = ""
                        if tail_1.Modify[0] == '!':
                                # 만약 ! ( 부정 )연산자 일경우 피연산자는 한개만 Pop합니다.
                                TempValue_1 = tail_2.Value
                                [top, tail_2] = Node.Pop(Operand)
                                TempModify += top.Modify

                                TempOp = tail_1.Modify[0]
                                [top, tail_1] = Node.Pop(Operator)
                                TempModify += top.Modify

                                ResultValue = MainOperator(TempValue_1, TempOp, TempValue_1)
                                tail_2 = Node.Push(Operand, ResultValue, TempModify)
                                TempModify = cSwap(TempModify)
                        else:
                                # 만약 ! ( negation )연산자가 아닐경우 피연산자는 두개를 Pop합니다.
                                TempValue_2 = tail_2.Value
                                [top, tail_2] = Node.Pop(Operand)
                                TempModify += top.Modify

                                TempOp = tail_1.Modify[0]
                                [top, tail_1] = Node.Pop(Operator)
                                TempModify += top.Modify

                                TempValue_1 = tail_2.Value
                                [top, tail_2] = Node.Pop(Operand)
                                TempModify += top.Modify

                                ResultValue = MainOperator(TempValue_1, TempOp, TempValue_2)
                                tail_2 = Node.Push(Operand, ResultValue, TempModify)
                                TempModify = cSwap(TempModify)
                        # 연산자의 Pop연산이 일어났기때문에 합성명제를 출력합니다.
                        if j == 0 and modifyPrint == 0:
                                w.write("    %s    |" % TempModify)
                                WholeLineLength += HowManyLength(TempModify)+7
                        else:
                                modifyLength = HowManyLength(TempModify)+6
                                for p in range(0, int(ceil(modifyLength/2))):
                                        w.write(" ")
                                w.write("%s" %('T' if tail_2.Value else 'F') )
                                for p in range(0, int(ceil((modifyLength-1)/2))):
                                        w.write(" ")
                                w.write("|")
                tail_1 = Node.Push(Operator, 0, InModify[i])

# 문자열 끝났을시 Operator에 남아있는 연산자들을 꺼내며 연산합니다.
while Operator.HeadArrive == 1:
        TempModify = ""
        if tail_1.Modify[0] == '!':
                # 만약 ! ( 부정 )연산자 일경우 피연산자는 한개만 Pop합니다.
                TempValue_1 = tail_2.Value
                [top, tail_2] = Node.Pop(Operand)
                TempModify += top.Modify

                TempOp = tail_1.Modify[0]
                [top, tail_1] = Node.Pop(Operator)
```

```
                TempModify += top.Modify

                ResultValue = MainOperator(TempValue_1, TempOp, TempValue_1)
                tail_2 = Node.Push(Operand, ResultValue, TempModify)
                TempModify = cSwap(TempModify)
            else:
                TempValue_2 = tail_2.Value
                [top, tail_2] = Node.Pop(Operand)
                TempModify += top.Modify

                TempOp = tail_1.Modify[0]
                [top, tail_1] = Node.Pop(Operator)
                TempModify += top.Modify

                TempValue_1 = tail_2.Value
                [top, tail_2] = Node.Pop(Operand)
                TempModify += top.Modify

                ResultValue = MainOperator(TempValue_1, TempOp, TempValue_2)
                tail_2 = Node.Push(Operand, ResultValue, TempModify)
                TempModify = cSwap(TempModify)
            # 연산자의 Pop연산이 일어났기때문에 합성명제를 출력합니다.
                if j == 0 and modifyPrint == 0:
                    w.write("    %s    |" % TempModify)
                    WholeLineLength += HowManyLength(TempModify)+7
                else:
                    modifyLength = HowManyLength(TempModify)+6
                    for p in range(0, int(ceil(modifyLength/2))):
                        w.write(" ")
                    w.write("%s" %('T' if tail_2.Value else 'F') )
                    for p in range(0, int(ceil((modifyLength-1)/2))):
                        w.write(" ")
                    w.write("|")
        Operand.HeadArrive = 0
        w.write("₩n")
        if j == 0 and modifyPrint == 0:
            j = -1
            modifyPrint = 1
            for p in range(0, WholeLineLength):
                w.write("=")
            w.write("₩n")
        j += 1
printf("₩n")
fp.close()
stdin.readline()
```