Programming Language Assignment #1

Internal Document

소프트웨어대학 소프트웨어학부

20204043 문벼리

```
code = open('/Users/PC/Desktop/PLhomework/code4.txt')  ⚠ 1  ⚠ 190  ✓ 26

#토큰코드 선언
ident = 1
const = 2
assign_op = 11
semi_colon = 12
add_op = 13
mult_op = 14
left_paren = 15
right_paren = 16

# 식별자, 연산자, 숫자의 개수 저장하는 변수
tokenDic={"ID":0,"CONST":0,"OP":0}
tokenDicList=[]
identDic={}
idConList=[]
tList=[]
lineList=[]

# 에러메시지를 리스트에 저장한 다음 한 번에 출력
error = []

```

먼저 코드를 읽어올 txt 파일을 open 했다.

프로그램 전역에서 쓰일 토큰 코드와 문장별 토큰의 개수를 저장할 딕셔너리(tokenDic), 토큰코드를 저장할 리스트(tList), 어휘를 저장할 리스트(idConList), 연산을 수행할때 사용할 딕셔너리(identDic), 컴파일 이후의 문장을 저장할 리스트(lineList), 에러 메시지를 저장할 리스트(error)을 선언했다.

```
25
26      def genWordList():
27          wordList = []
28          codeline=code.read().split('\n')
29          for line in codeline:
30              line=line.split(' ')
31              list=[i for i in line if i]
32              wordList.append(list)
33          #공백 제거
34          return wordList
35
```

전체 코드를 띄어쓰기 단위로 파싱해주는 genWordList()함수이다. 공백이 하나 이상일 때도 공백을 모두 제거하도록 I for I in line if I 구문을 사용했다.

```
# 단순어휘분석기
def lex(list):
    wordList=list
    for line in wordList:
        for word in line:
            #ident 인지 확인
            idConList.append(word)
            idCon(word)
        tokenDicList.append(tokenDic.copy())
    return
```

파싱된 리스트를 받아서 idCon()함수에 순서대로 넣어주는 lex()함수이다.

```
def idCon(string):
    if string[0].isalpha()==True or string[0]=='_':
        for char in string:
            if char.isalnum()==True or char=='_':
                continue
            else:
                lookup(string)
                return
        tList.append(ident)
        identDic[string] = 0
    elif string.isdigit()==True:
        tList.append(const)
    else:
        lookup(string)
        return
```

string 을 매개변수로 받아 토큰 리스트에 토큰을 append 해주는 idCon()함수이다. 위에서 선언했던 토큰 코드를 append 해준다.

```
def lookup(string):
    if string == ':=' or string=='=' or string==':':
        tList.append(assign_op)
    elif string == ';':
        tList.append(semi_colon)
    elif string == '+' or string == '-':
        tList.append(add_op)
    elif string == '*' or string == '/':
        tList.append(mult_op)
    elif string == '(':
        tList.append(left_paren)
    elif string == ')':
        tList.append(right_paren)
    else:
        for i in string:
            if ord(i) <= 32:
                stringList = string.split(i)
                if idConList:
                    idConList.pop(-1)
                idConList.extend(stringList)
                print(stringList)
                for word in stringList:
                    print(word)
                    idCon(word)
                return
            elif i in ['(', ')', '+', '-', '*', '/', ';']:
                stringList = string.split(i)
                string=str(' '+i+' ').join(stringList).strip()
                stringList = string.split(' ')
                if idConList:
                    idConList.pop(-1)
                idConList.extend(stringList)
                for word in stringList:
                    idCon(word)
                return
            elif i in [':','=']:
                if ':=' in string:
                    stringList = string.split(':=')
                    string = str(' := ').join(stringList).strip()
                    stringList = string.split(' ')
                    if idConList:
                        idConList.pop(-1)
                    idConList.extend(stringList)
                    for word in stringList:
                        idCon(word)
                    return
                else:
                    stringList = string.split(i)
                    string = str(' '+i+' ').join(stringList).strip()
                    stringList = string.split(' ')
                    if idConList:
                        idConList.pop(-1)
                    idConList.extend(stringList)
                    for word in stringList:
                        idCon(word)
                    return
    return
```

ident 와 const 를 제외한 토큰들을 append 할 때 사용하는 lookup()함수이다. Assign op 는 두 글자 중 한 글자만 작성했더라도 정상적으로 파싱되도록 했다. (나중에 파싱 테이블에서 warning 오류가 뜨도록 했다.) lookup 함수에서 혹시 띄어쓰기가 안되었더라도 괄호나 연산자가 있으면 괄호나 연산자 기준으로 일단 파싱하도록 했다. 이 때에도 만약에 assign op 에서 두 글자 중 한 글자만 작성했더라도 정상적으로 파싱되도록 했다.

```
def factor(list,value):
    if list[0]==left_paren:
        list.pop(0)
        lineList.append(idConList.pop(0))
        if list[0]==left_paren or list[0]==ident or list[0]==const:
            list,value=expression(list,value)
            if not list:
                lineList.append(')')
                error.append("(Warning) 닫는 괄호 추가")
            elif list[0]==right_paren:
                list.pop(0)
                lineList.append(idConList.pop(0))
            elif list[0]==left_paren:
                list.pop(0)
                idConList.pop(0)
                lineList.append(')')
                error.append("(Warning) 괄호 방향 변경")
        elif list[0]==right_paren:
            if list[1]==ident or list[1]==const or list[1]==left_paren:
                error.append("(Warning) 괄호 제거: )")
                list.pop(0)
                idConList.pop(0)
            else:
                error.append("(Error) 해당 괄호에 맞는 left_paren 을 찾을 수
없습니다.")
                list.pop(0)
                idConList.pop(0)
                list,value=factor(list,value)
        elif list[0]==assign_op or list[0]==semi_colon:
            error.append("(Warning) 한 문장에 하나만 사용 가능합니다:
"+idConList.pop(0))
            list.pop(0)
            list,value=factor(list,value)
    elif list[0]==ident:
        tokenDic["ID"] += 1
        list.pop(0)
        if identDic[idConList[0]] :
            lineList.append(idConList[0])
            value=identDic[idConList.pop(0)]
        else:
            error.append("(Error) 정의되지 않은 변수(" +idConList[0]+")가
참조됨")
```

```
            identDic[idConList.pop(0)]='Unknown'
    elif list[0]==const:
        tokenDic["CONST"]+=1
        list.pop(0)
        lineList.append(idConList[0])
        value=int(idConList.pop(0))
    elif list[0]==add_op or list[0]==mult_op:
        error.append("(Warning) 중복 연산자 제거: "+idConList.pop(0))
        list.pop(0)
        list,value=factor(list,value)
    else:
        error.append("(Error) "+idConList.pop(0)+"을 인식할 수 없습니다")
        list.pop(0)
    return list,value
```

factor()함수이다. 최대한 예외처리를 해주었지만 인식 안되는 토큰에 대해서는 에러를 반환한다. 연산자가 중복해서 등장한다면 제일 먼저 나온 연산자가 아닌 다른 연산자는 모두 무시하고, 괄호의 방향이 반대로 되었거나 괄호가 없다면 괄호를 추가해서 파싱해준다.

```
def factorTail(list,value):
    if not list:
        return list,value
    elif list[0]==mult_op:
        list.pop(0)
        lineList.append(idConList[0])
        op=idConList.pop(0)
        list,temp=factor(list,value)
        tokenDic["OP"] += 1
        if value!="Unknown" and temp!="Unknown":
            if op=='*':
                value*=temp
            else:
                value/=temp
            list, value = factorTail(list, value)
        else:
            value="Unknown"
    elif list[0]==right_paren or list[0]==left_paren:
        if list[1]==mult_op:
            list.pop(0)
            print(idConList[0])
            error.append("(Warning) 괄호 오류 제거: ",idConList.pop(0))
            list, value=factorTail(list,value)
    return list,value
```

factortail()함수이다. 연산자 말고 괄호가 등장한다면 괄호를 제거해 준다.

```
def term(list,value):
    list,value=factor(list,value)
    list,value=factorTail(list,value)
    return list,value
```

term()함수이다.

```
def termTail(list,value):
    if not list:
        return list,value
    elif list[0] == add_op:
        list.pop(0)
        lineList.append(idConList[0])
        op=idConList.pop(0)
        list,temp=term(list,value)
        tokenDic["OP"] += 1
        if value!="Unknown" and temp!="Unknown":
            if op=='+':
                value+=temp
            else:
                value-=temp
            list,value=termTail(list,value)
        else:
            value="Unknown"
    elif list[0]==right_paren or list[0]==left_paren:
        if list[1]==add_op:
            list.pop(0)
            error.append("(Warning) 괄호 오류 제거: "+idConList.pop(0))
            list, value=termTail(list,value)
    return list,value
```

termTail()함수이다. 연산자가 아닌 괄호가 등장한다면 괄호를 제거해준다.

```
def expression(list,value):
    list,value=term(list,value)
    list,value=termTail(list,value)
    return list,value
```

expression()함수이다.

```
def statement(list):
    if list[0] == ident:
        list.pop(0)
        tokenDic["ID"]+=1
        lineList.append(idConList[0])
        id=idConList.pop(0)
        if list[0] == assign_op:
            list.pop(0)
            if idConList[0] == ':' or idConList[0] == '=':
                lineList.append(':=')
                error.append("(Warning) " + idConList[0] + "을 :=으로 변경")
            else:
                lineList.append(idConList[0])
            idConList.pop(0)
            value='Unknown'
            list,value=expression(list,value)
            identDic[id]=value
```

```
        elif list[1]==assign_op:
            error.append("(Warning) 인식할 수 없는 토큰 삭제:
"+idConList.pop(0))
            list.pop(0)
            list.pop(0)
            if idConList[0] == ':' or idConList[0] == '=':
                lineList.append(':=')
                error.append("(Warning) " + idConList[0] + "을 :=으로 변경")
            else:
                lineList.append(idConList[0])
            idConList.pop(0)
            value = 'Unknown'
            list, value = expression(list, value)
            identDic[id] = value
        elif list[0]==ident:
            error.append("(Warning) assign_op 가 필요합니다.")
            lineList.append(':=')
            idConList.pop(0)
            value = 'Unknown'
            list, value = expression(list, value)
            identDic[id] = value
        else:
            error.append("(Error) assign_op 가 없습니다.")
    else:
        error.append("(Error) ident 가 없습니다.")
    return list
```

statement()함수이다. Assign op 가 두 글자 중 한 글자만 있다면 정상적으로 파싱해주고, 인식할 수 없는 토큰이 있다면 제거하고 파싱해준다. Assign op 가 없는 경우, 바로 뒤에 ident 가 있으면 추가해주고 파싱하고, 없다면 에러 메시지를 반환한다. Ident 로 시작하지 않는 경우에도 에러를 반환한다.

```
def statements(tokenList):
    tokenList=statement(tokenList)
    if not tokenList:
        print(' '.join(lineList))
        print(tokenDic)
        if error:
            for line in error:
                print(line)
        else:
            print("(OK)")
        print("Result==>",identDic)
    elif tokenList[0] == semi_colon:
        tokenList.pop(0)
        lineList.append(idConList[0])
        idConList.pop(0)
        print(' '.join(lineList))
        lineList.clear()
        print(tokenDic)
        tokenDic["ID"] = 0
        tokenDic["CONST"] = 0
        tokenDic["OP"] = 0
```

```
        if error:
            for line in error:
                print(line)
            error.clear()
        else:
            print("(OK)")
        tokenList=statements(tokenList)
    elif tokenList[0]==ident:
        error.append("(Warning) statement와 statement 사이에는 semi_colon이
필요합니다.")
        lineList.append(';')
        print(' '.join(lineList))
        lineList.clear()
        print(tokenDic)
        tokenDic["ID"] = 0
        tokenDic["CONST"] = 0
        tokenDic["OP"] = 0
        if error:
            for line in error:
                print(line)
            error.clear()
        else:
            print("(OK)")
        tokenList = statements(tokenList)
    else:
        while tokenList[0]!=semi_colon:
            if not tokenList:
                error.append("(Error) 올바르지 않은 문장입니다.")
                return tokenList
            lineList.append(idConList.pop(0))
            tokenList.pop(0)
        error.clear()
        error.append("(Error) 올바르지 않은 문장입니다.")
        tokenList.pop(0)
        lineList.append(idConList[0])
        idConList.pop(0)
        print(' '.join(lineList))
        lineList.clear()
        print(tokenDic)
        tokenDic["ID"] = 0
        tokenDic["CONST"] = 0
        tokenDic["OP"] = 0
        if error:
            for line in error:
                print(line)
            error.clear()
        else:
            print("(OK)")
        tokenList = statements(tokenList)
    return tokenList
```

statements()함수이다. 세미콜론과 세미콜론 사이에서 파싱된 문장, 토큰딕셔너리, 에러리스트를 print 해준다. 마지막 문장에서는 result 를 print 해준다. 만약 세미콜론이 없고, 다음 문장이 ident 로 시작한다면 세미콜론을 추가해준다.

```
wordlist=genWordList()
lex(wordlist)
statements(tList)
```

메인 코드이다.