编译原理课程设计
实验报告

指导教师：张鹏

年　　级：2019 级

班　　级：23 班

小组编号：　5

组长学号姓名：21191511 张轶博

组员学号姓名：09190717 邓秋怡

组员学号姓名：21191129 李林峰

2022 年　　4 月　　27 日

计算机科学与技术学院

完成实验内容

小组实现了词法分析模块、递归下降语法分析模块、LL1 语法分析模块、语义分析模块四个核心模块，使用 pyecharts 对程序产物语法树进行了良好的可视化。

进一步，我们使用 pyqt5 设计、完成了 SNL 语言分析程序的交互界面，集成了程序输入、控制台信息和所有程序产物，提升了使用体验。

小组成员任务完成情况

姓名	具体完成任务	工作量百分比
张轶博	完成词法分析、语义分析、SNL 可视化界面、部分语法树可视化代码；整合各个子模块为完整系统	35%
邓秋怡	完成 PREDICT 集的生成，递归下降语法分析、语法错误检查。	35%
李林峰	完成 LL1 语法分析、语法错误检查和部分语法树可视化代码	30%

小组成员协作情况

使用 github 实现代码同步，在实验进行中多次开会讨论实验细节，小组成员互帮互助，共同攻克遇到的难题。

实验平台与编程语言

词法分析、PREDICT 集的生成、LL1 语法分析、语义分析、可视化采用 Python 完成，递归下降语法分析采用 C++ 完成。相关库版本如下：

```
pyqt          5.9.2
pyecharts     1.9.1
```

实验方案设计

PREDICT 集产生：

根据文法的文法规则，按顺序生成了 first 集、follow 集、predict 集，对教材上的 predict 集进行了验证和一定的修改，同时产生 predict 集的程序可以使得我们的语法分析部分程序更加灵活。当对文法规则进行修改或者增加删除时，只需要对文法规则的文本进行增删改 就能做到动态生成 predict 集，不必修改语法分析部分的程序。

注 1：因为发现在原文法规则下，对于字符来说没有相应的指代符，即非终极符 Exp 无法推出 CHARC，所以我们对文法加了一条 Factor::=CHARC 的文法。

```
105    Factor ::= CHARC
```

注 2：产生后对书上的 predict 集的校验结果如下：

①第 48 条文法规则：ParamMore ::= NULL 的 predict 集应该改为{')'}

```
46 {'RECORD', 'INTEGER', 'CHAR', 'ID', 'VAR', 'ARRAY'}
47 {'RECORD', 'INTEGER', 'CHAR', 'ID', 'VAR', 'ARRAY'}
48 {')'}
50 {'RECORD', 'INTEGER', 'CHAR', 'ID', 'ARRAY'}
```

46	{ INTEGER,CHAR,ARRAY, RECORD,ID ,VAR }
47	{ INTEGER,CHAR,ARRAY, RECORD, ID, VAR }
48	{ (}
49	{ ; }
50	{ INTEGER,CHAR,ARRAY, RECORD,ID }

②第 67 条文法规则：AssCall ::= AssignmentRest 的 predict 集应该改为{':=', '.', '['}

```
65 {'RETURN'}
66 {'ID'}
67 {':=', '.', '['}
68 {'('}
```

65	{ RETURN }
66	{ ID }
67	{ := }
68	{ (}

③第 94 条文法规则：VariMore ::= NULL 的 predict 集应该改为{'THEN', ')', '-', 'FI', ';', '<', '=', 'DO', ']', '*', ',', 'ELSE', ':=', '/', '+', 'ENDWH', 'END'}

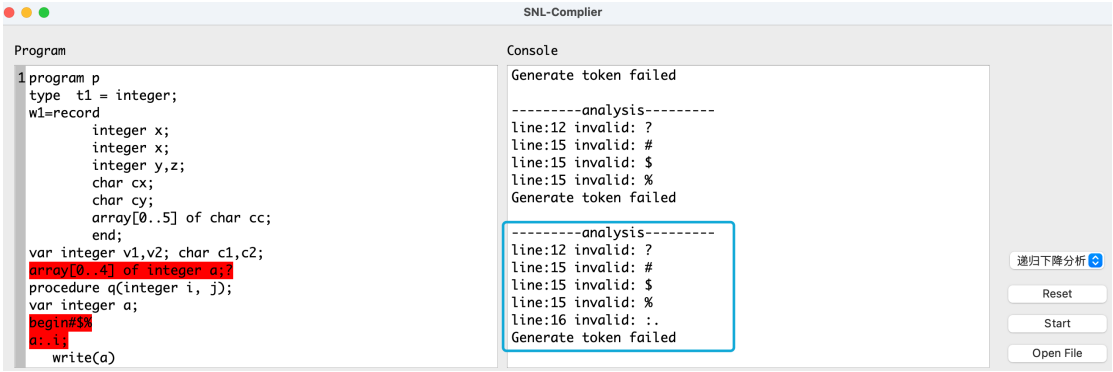
```
93 {'THEN', ')', '-', 'FI', ';', '<', '=', 'DO', ']', '*', ',', 'ELSE', ':=', '/', '+', 'ENDWH', 'END'}
```

93	{ :=, * , / , + , - , <, =,THEN, ELSE, FI, DO, ENDWH,),END, ; ,COMMA }

④增加 Factor::=CHARC 的文法后，文法规则 78，86，83，81 的 predict 都会比书上多一个 CHARC。

词法分析：

预先生成保留字，运算符以及限界符，以教材上的状态 DFA 作为参考，每次读入一个 token，并根据状态 DFA 进行非法判断及状态转移。当出现异常字符，或非法状态时，程序会抛出错误详细信息。若词法分析顺利完成，会生成 token 序列结果文件，供语义分析进行使用。



LL1 语法分析：

由 LL1 驱动程序、语法树搭建、语法错误检测三部分组成，用户输入词法分析程序产生的 token 序列结果文件，经过 LL1 分析后输出语法树文件和语法报错信息。

① LL1 驱动程序由 token 序列和符号栈依据 LL1 分析表进行替换、匹配、接受、报错等步骤。其中替换、匹配、接受三步需运行语法树搭建，报错则需运行语法错误检测。

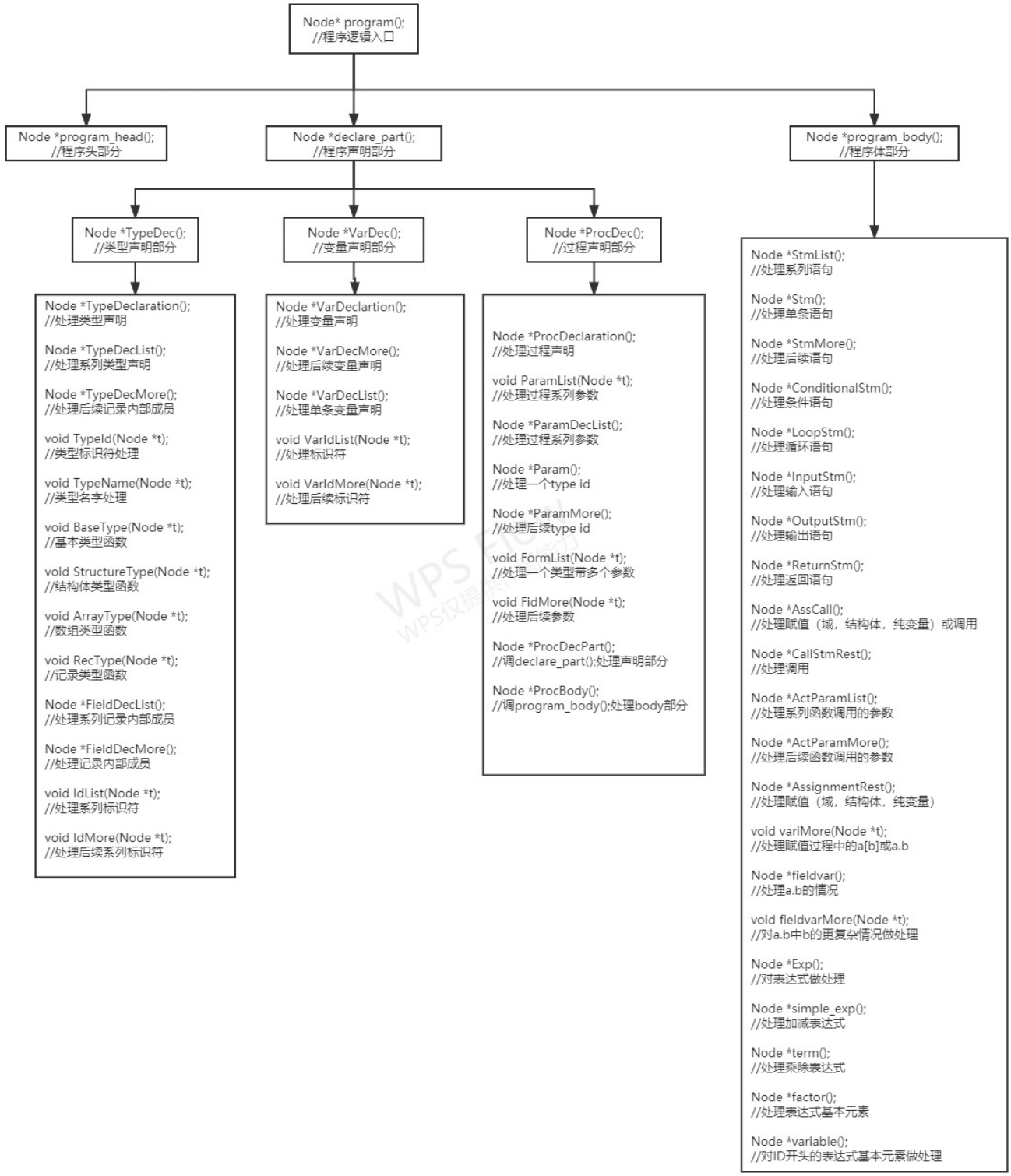
② 由于语法树搭建部分操作复杂，我们对每一条文法分别编写执行函数，当 LL1 驱动程序判断执行的具体文法之后，调用该条文法相应的语法树搭建函数。此过程中共用到三个数据栈：语法树栈、操作符栈、操作数栈。

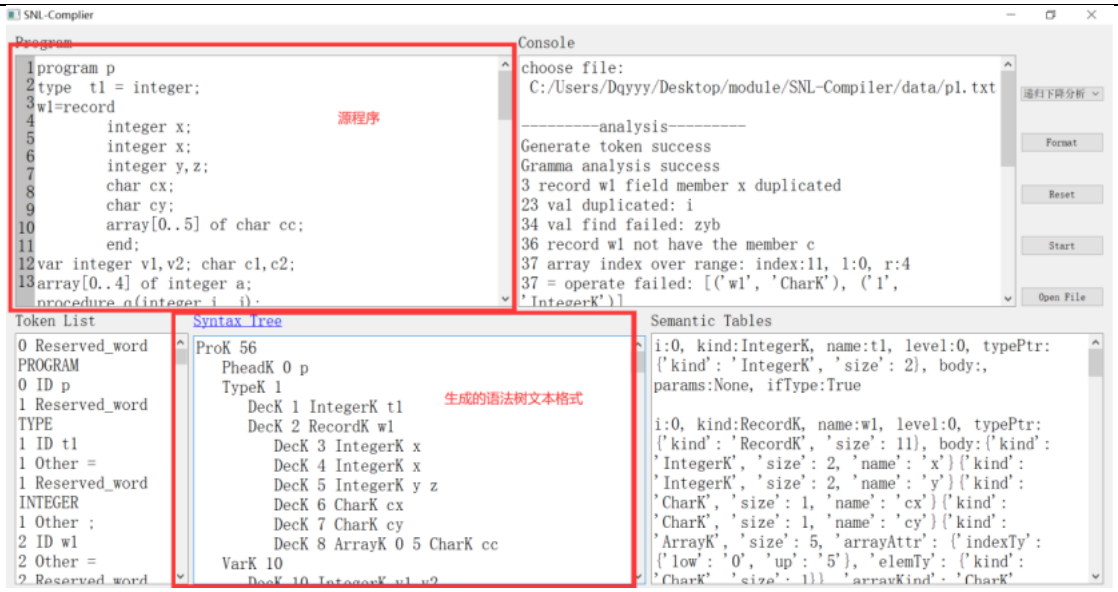
③ 语法错误检测部分会在 LL1 驱动程序执行报错步骤后运行，对该语法错误种类进行识别，并在识别成功之后尝试对符号栈和 token 序列进行修复。若能成功修复则 LL1 驱动程序可继续运行检测有无其他语法错误。

递归下降语法分析：

整体思想是每个非终极符和函数一一对应，根据文法和当前输入符号，利用 predict 集确定一条文法，然后调用新的非终极符对应的函数继续往下递归。用户输入词法分析程序产生的 token 序列结果文件，经过递归下降分析后输出语法树文件和语法报错信息，如果遇到错误会跳过然后继续分析后面的 token，直到分析到文件尾。

程序逻辑框图如下：





语义分析：

从语法分析部分得到运行产物语法树文件。通过分析语法树文件，在内存中建立包含所需信息的语法树，再通过 dfs 该语法树，在判断语义错误的同时生成符号表。过程中发现语义错误，则抛出错误详细信息。

程序输出符号表文件。

支持的语义错误如下：

（1） 标识符的重复定义；

（2） 无声明的标识符；

（3） 标识符为非期望的标识符类别（类型标识符，变量标识符，过程名标识符）；

（4） 数组类型下标越界错误；

（5） 数组成员变量和域变量的引用不合法；

（6） 赋值语句的左右两边类型不相容；

（7） 赋值语句左端不是变量标识符；

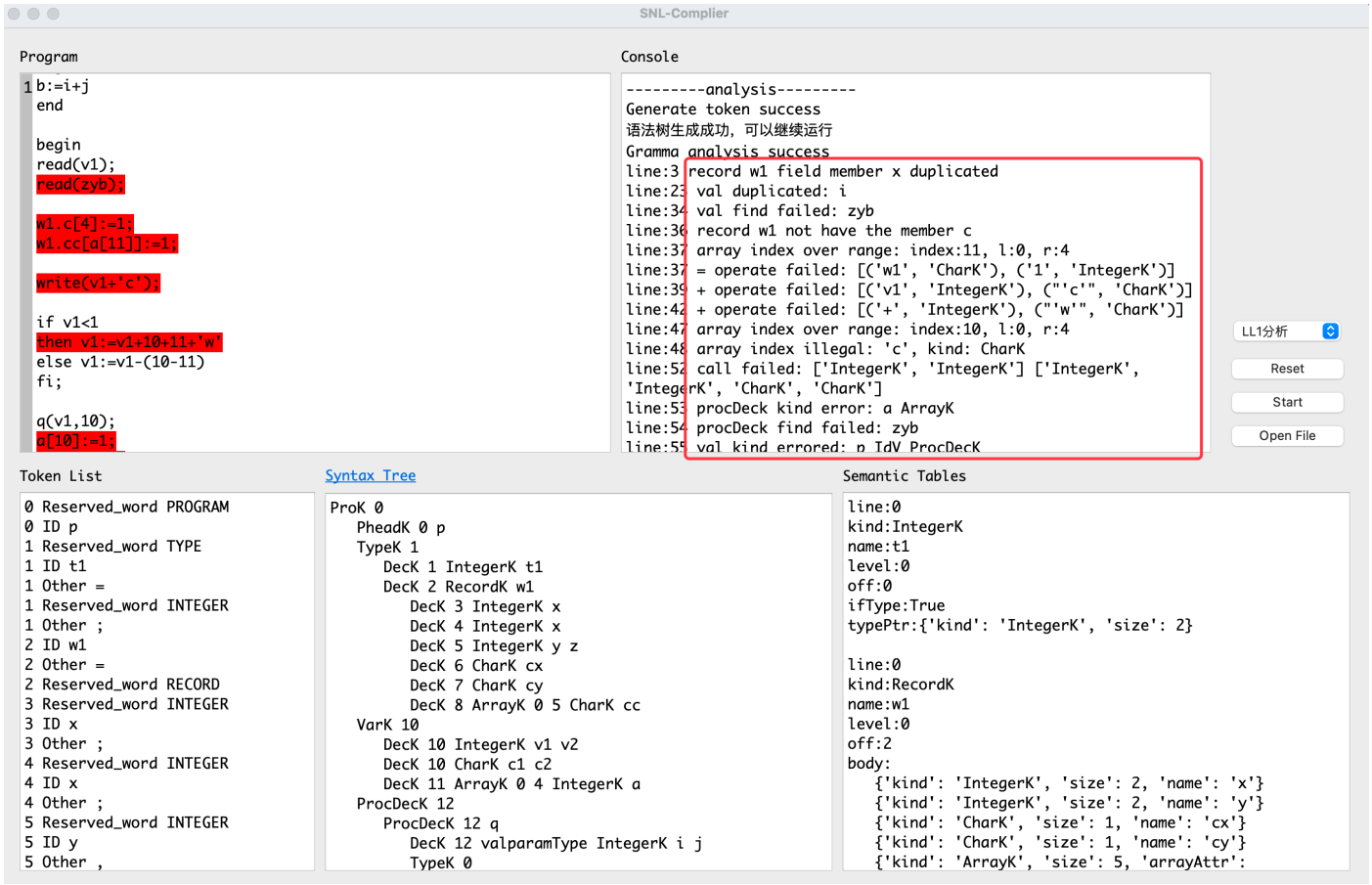
（8） 过程调用中 ， 形实参类型不匹配；

（9） 过程调用中， 形实参个数不相同；

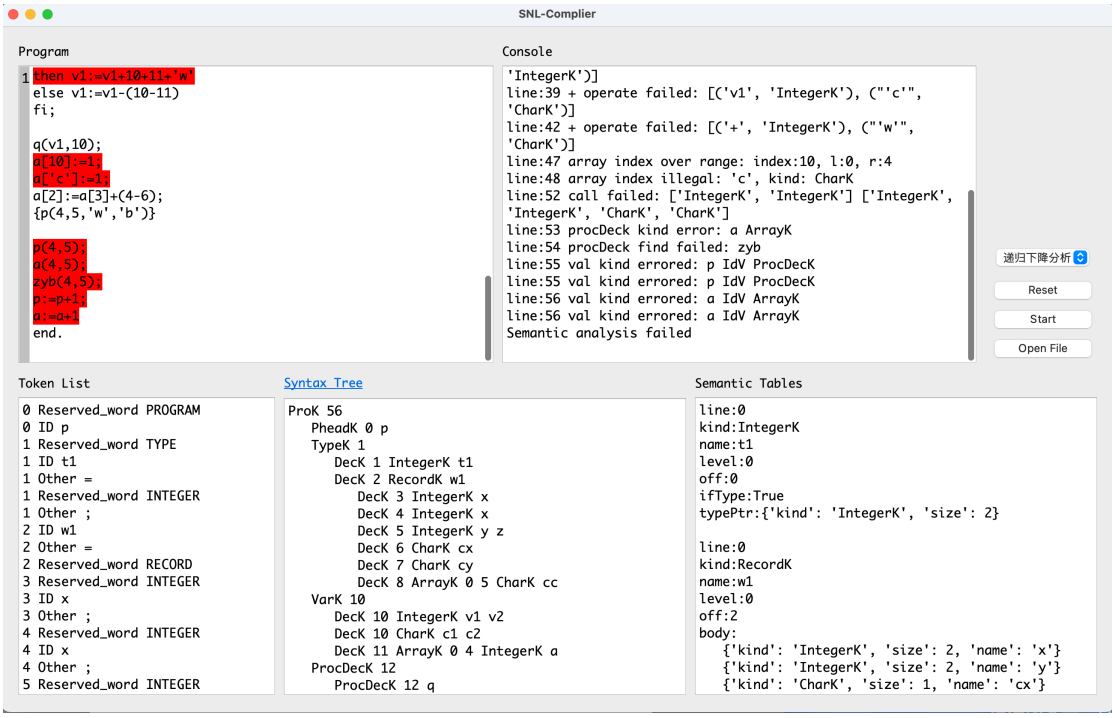
（10）过程调用语句中，标识符不是过程标识符；

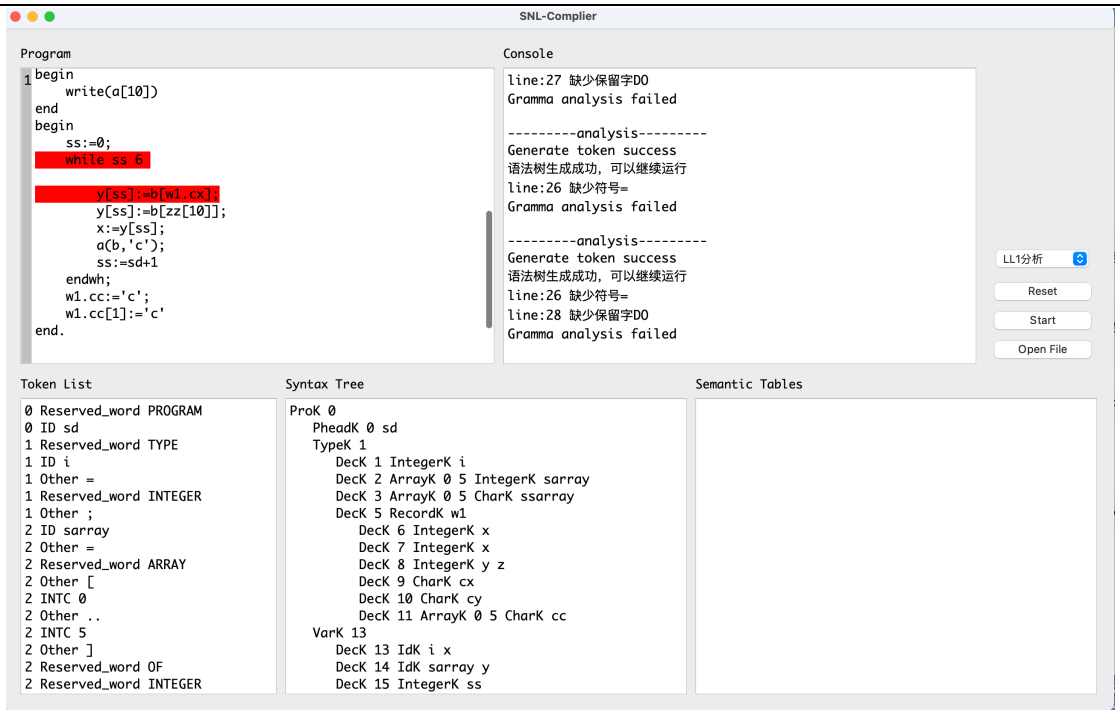
（11）if 和 while 语句的条件部分不是 bool 类型；

（12）表达式中运算符的分量的类型不相容

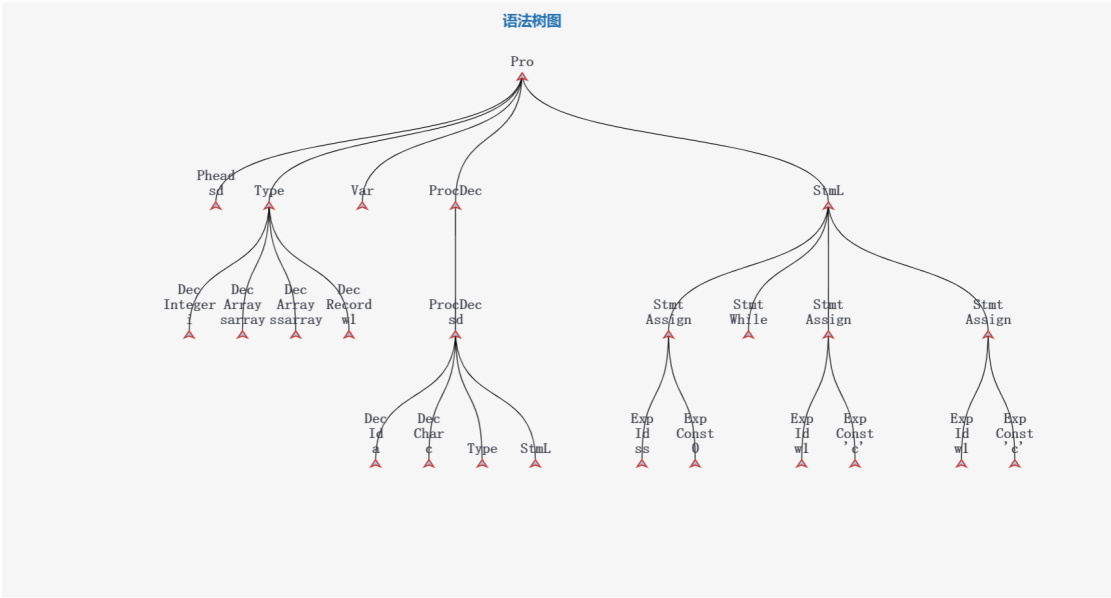

SNL-Complier
Program
1 b:=i+j
end
begin
read(v1);
read(zyb);
w1.c[4]:=1;
w1.cc[a[11]]:=1;
write(v1+'c');
if v1<1
then v1:=v1+10+11+'w'
else v1:=v1-(10-11)
fi;
q(v1,10);
a[10]:=1;
Console
---------analysis---------
Generate token success
语法树生成成功，可以继续运行
Gramma analysis success
line:3 record w1 field member x duplicated
line:23 val duplicated: i
line:34 val find failed: zyb
line:36 record w1 not have the member c
line:37 array index over range: index:11, l:0, r:4
line:37 = operate failed: [('w1', 'CharK'), ('1', 'IntegerK')]
line:39 + operate failed: [('v1', 'IntegerK'), ("'c'", 'CharK')]
line:42 + operate failed: [('+', 'IntegerK'), ("'w'", 'CharK')]
line:47 array index over range: index:10, l:0, r:4
line:48 array index illegal: 'c', kind: CharK
line:52 call failed: ['IntegerK', 'IntegerK'] ['IntegerK',
'IntegerK', 'CharK', 'CharK']
line:53 procDeck kind error: a ArrayK
line:54 procDeck find failed: zyb
line:55 val kind errored: p IdV ProcDecK
LL1分析
Reset
Start
Open File
Token List
0 Reserved_word PROGRAM
0 ID p
1 Reserved_word TYPE
1 ID t1
1 Other =
1 Reserved_word INTEGER
1 Other ;
2 ID w1
2 Other =
2 Reserved_word RECORD
3 Reserved_word INTEGER
3 ID x
3 Other ;
4 Reserved_word INTEGER
4 ID x
4 Other ;
5 Reserved_word INTEGER
5 ID y
5 Other ,
Syntax Tree
ProK 0
PheadK 0 p
TypeK 1
DecK 1 IntegerK t1
DecK 2 RecordK w1
DecK 3 IntegerK x
DecK 4 IntegerK x
DecK 5 IntegerK y z
DecK 6 CharK cx
DecK 7 CharK cy
DecK 8 ArrayK 0 5 CharK cc
VarK 10
DecK 10 IntegerK v1 v2
DecK 10 CharK c1 c2
DecK 11 ArrayK 0 4 IntegerK a
ProcDecK 12
ProcDecK 12 q
DecK 12 valparamType IntegerK i j
TypeK 0
Semantic Tables
line:0
kind:IntegerK
name:t1
level:0
off:0
ifType:True
typePtr:{'kind': 'IntegerK', 'size': 2}
line:0
kind:RecordK
name:w1
level:0
off:2
body:
{'kind': 'IntegerK', 'size': 2, 'name': 'x'}
{'kind': 'IntegerK', 'size': 2, 'name': 'y'}
{'kind': 'CharK', 'size': 1, 'name': 'cx'}
{'kind': 'CharK', 'size': 1, 'name': 'cy'}
{'kind': 'ArrayK', 'size': 5, 'arrayAttr':

程序界面及运行截图


SNL-Complier
Program
1 then v1:=v1+10+11+'w'
else v1:=v1-(10-11)
fi;
q(v1,10);
a[10]:=1;
a['c']:=1;
a[2]:=a[3]+(4-6);
{p(4,5,'w','b')}
p(4,5);
a(4,5);
zyb(4,5);
p:=p+1;
a:=a+1
end.
Console
'IntegerK')]
line:39 + operate failed: [('v1', 'IntegerK'), ("'c'",
'CharK')]
line:42 + operate failed: [('+', 'IntegerK'), ("'w'",
'CharK')]
line:47 array index over range: index:10, l:0, r:4
line:48 array index illegal: 'c', kind: CharK
line:52 call failed: ['IntegerK', 'IntegerK'] ['IntegerK',
'IntegerK', 'CharK', 'CharK']
line:53 procDeck kind error: a ArrayK
line:54 procDeck find failed: zyb
line:55 val kind errored: p IdV ProcDecK
line:55 val kind errored: p IdV ProcDecK
line:56 val kind errored: a IdV ArrayK
line:56 val kind errored: a IdV ArrayK
Semantic analysis failed
递归下降分析
Reset
Start
Open File
Token List
0 Reserved_word PROGRAM
0 ID p
1 Reserved_word TYPE
1 ID t1
1 Other =
1 Reserved_word INTEGER
1 Other ;
2 ID w1
2 Other =
2 Reserved_word RECORD
3 Reserved_word INTEGER
3 ID x
3 Other ;
4 Reserved_word INTEGER
4 ID x
4 Other ;
5 Reserved_word INTEGER
Syntax Tree
ProK 56
PheadK 0 p
TypeK 1
DecK 1 IntegerK t1
DecK 2 RecordK w1
DecK 3 IntegerK x
DecK 4 IntegerK x
DecK 5 IntegerK y z
DecK 6 CharK cx
DecK 7 CharK cy
DecK 8 ArrayK 0 5 CharK cc
VarK 10
DecK 10 IntegerK v1 v2
DecK 10 CharK c1 c2
DecK 11 ArrayK 0 4 IntegerK a
ProcDecK 12
ProcDecK 12 q
Semantic Tables
line:0
kind:IntegerK
name:t1
level:0
off:0
ifType:True
typePtr:{'kind': 'IntegerK', 'size': 2}
line:0
kind:RecordK
name:w1
level:0
off:2
body:
{'kind': 'IntegerK', 'size': 2, 'name': 'x'}
{'kind': 'IntegerK', 'size': 2, 'name': 'y'}
{'kind': 'CharK', 'size': 1, 'name': 'cx'}



语法树可视化界面：



源程序核心代码

产生 PREDICT 集的核心代码：

```
import copy

arr = []
left = set()
right = set()
first = {"": set()}
follow = {"": set()}
predict = {0: set()}

def f(x, only_right):
    i = 0
```

```
    flag = 0
    for i in range(2, len(x)):  # 遍历右边的串
        if x[i] in only_right:  # 遇到终极符了
            first[x[0]].add(x[i])
            flag = 1
            break
        elif "NULL" not in first[x[i]]:  # 都非空了
            first[x[0]] = first[x[0]].union(first[x[i]])
            flag = 1
            break
        else:  # 还没到终极符并且有非空
            first[x[0]] = first[x[0]].union(first[x[i]]) - {"NULL"}
    if flag == 0 and ("NULL" in first[x[len(x) - 1]]):
        first[x[0]].add("NULL")

def h(x, i, only_right):
    j = i + 1
    while j < len(x) and (x[j] not in only_right) and ("NULL" in first[x[j]]):
        # 退出：j 超了，是终极符，非终但是没有 null
        follow[x[i]] = follow[x[i]].union(first[x[j]]) - {"NULL"}
        j = j + 1
    if (j == len(x)):
        follow[x[i]] = follow[x[i]].union(follow[x[0]])
    elif (x[j] in only_right):
        follow[x[i]].add(x[j])
    else:
        follow[x[i]] = follow[x[i]].union(first[x[j]])

def p(x, i, only_right):  # i 是行号，x 是行
    j = 2
    while j < len(x) and (x[j] not in only_right) and ("NULL" in first[x[j]]):
        # 退出：j 超了，是终极符，非终但是没有 null
        predict[i] = predict[i].union(first[x[j]]) - {"NULL"}
        j = j + 1
    if j == len(x):  # 超过了
        predict[i] = predict[i].union(follow[x[0]])
    elif x[j] in only_right and x[j] != "NULL":  # 非空外的终极符
        predict[i].add(x[j])
    elif x[j] in only_right and x[j] == "NULL":  # 是空的终极符
        predict[i] = predict[i].union(follow[x[0]])
    else:  # 全部没有 Null
        predict[i] = predict[i].union(first[x[j]])


def getPredict():
    with open("../data/grammar.txt") as file:
        lines = file.readlines()
        for line in lines:  # 得到 left 和 right
            line = str(line).replace("\n", "")
```

```
            pos = line.split(" ", 20)
            arr.append(pos)
            left.add(pos[0])  # left
            for x in pos[2:]:  # right
                right.add(x)
        only_right = right - left  # 只出现的右边的终极符

        for x in arr:  # 把一眼得到的 first 加进去
            if x[0] not in first.keys():  # 过了以后就都有关键字了
                first.update({x[0]: set()})
                follow.update({x[0]: set()})
            if x[2] in only_right:  # 右边第一个是终极符
                first[x[0]].add(x[2])
        t = copy.copy(first)
        while True:
            for y in arr:
                if y[2] not in only_right:
                    f(y, only_right)
            if t == first:
                break
            t = copy.copy(first)

        follow.update({arr[0][0]: {"#"}})
        t = copy.copy(follow)
        while True:
            for x in arr:
                for i in range(2, len(x)):
                    if x[i] not in follow.keys() and x[i] not in only_right:  # 还没有关键词并且需要创建关键词
                        follow.update({x[i]: set()})
                    if x[i] not in only_right:  # 只对非终极符进行函数调用
                        h(x, i, only_right)
            if t == follow:
                break
            t = copy.copy(follow)
        k = 1
        t = copy.copy(predict)
        while True:
            for x in arr:
                if k not in follow.keys():
                    predict.update({k: set()})
                p(x, k, only_right)
                k = k + 1
            if t == predict:
                break
            t = copy.copy(predict)
            k = 1
        print(first)
        print(follow)
```

```
        for key in predict:
            print(key, predict[key])

        # return predict, left, only_right
if __name__ == '__main__':
    getPredict();
```

词法分析核心代码：

```
import os

from config.config import delimiters, reservedWords

class Token:
    def __init__(self, line, lex, sem):
        self.line = line
        self.lex = lex
        self.sem = sem

tokenList = []
flag = 0

def init():
    global tokenList, flag
    tokenList = []
    flag = 0

def add(word, num, err=False):
    global flag
    if err:
        flag = -1
        tokenList.append(Token(num, "ERROR", word))
        print(f"line:{num + 1} invalid: {word}")
    elif str.isdigit(word):
        tokenList.append(Token(num, "INTC", int(word, 10)))
    elif word in delimiters:
        tokenList.append((Token(num, delimiters[word], word)))
    elif word in reservedWords:
        tokenList.append((Token(num, reservedWords[word], word)))
    elif word[0] == '\'' and word[-1] == '\'':
        tokenList.append((Token(num, "CHARC", word)))
    else:
        tokenList.append((Token(num, "ID", word)))

def work(lines):
    commentflag = False
    for num in range(0, len(lines)):
        line = lines[num].replace("\n", "", -1) + " "
        i = 0
        while i < len(line):
```

```
        c = line[i]
        if commentflag:
            if c == '}':
                commentflag = False
        elif str.isdigit(c):
            word = c
            while str.isdigit(line[i + 1]):
                word = word + line[i + 1]
                i = i + 1
            add(word, num)
        elif str.isalpha(c):
            word = c
            while str.isdigit(line[i + 1]) or str.isalpha(line[i + 1]):
                word = word + line[i + 1]
                i = i + 1
            add(word, num)
        elif c == '.':
            if line[i + 1] == ".":
                i = i + 1
                add("..", num)
            else:
                add(".", num)
        elif c == '\'':
            word = c
            i = i + 1
            while i < len(line):
                word = word + line[i]
                if line[i] == '\'':
                    add(word, num)
                    break
                elif (str.isdigit(line[i]) or str.isalpha(line[i])) == False:
                    add(word, num, True)
                    break
                i = i + 1
        elif c == '{':
            commentflag = True
        elif c == ':':
            if line[i + 1] == "=":
                add(":=", num)
            else:
                add(line[i] + line[i + 1], num, True)
            i = i + 1
        elif c in delimiters:
            add(c, num)
        elif c == " " or c == "":
            _ = c
        else:
            add(line[i], num, True)
        i = i + 1
```

```
    tokenList.append(Token(len(lines), "EOF", "EOF"))
    return tokenList

def lex(pro_path, token_path):
    init()
    if not os.path.exists(pro_path):
        print(f"Open pro_path:{pro_path} failed")
        return -1
    with open(pro_path) as file:
        lines = file.readlines()
        work(lines)
        # print(f"line: {x.line}, lex: {x.lex}, sem: {x.sem}")

    with open(token_path, "w") as file:
        for x in tokenList:
            if x.sem in delimiters:
                file.write(f"{x.line} Other {x.sem}\n")
            elif x.sem in reservedWords:
                file.write(f"{x.line} Reserved_word {x.lex}\n")
            else:
                file.write(f"{x.line} {x.lex} {x.sem}\n")
    if flag == 0:
        print("Generate token success")
    else:
        print("Generate token failed")
    return flag
```

LL1 语法分析核心代码：

```
# LL1 驱动程序
def run(self):
    syntax_tree = Tree()
    PreNode = syntax_tree.root
    while not self.SignStack.isEmpty() and self.TokenStack.peek()[2] != 'EOF':
        sign = self.SignStack.peek()
        toke = self.TokenStack.peek()
        if toke[1] == 'ID':
            token = 'ID'
        elif toke[1] == 'INTC':
            token = 'INTC'
        elif toke[1] == 'CHARC':
            token = 'CHARC'
        else:
            token = toke[2]
        if sign in self.left:  # 如果是非终极符，则用语法进行替换
            row = self.table_row[sign]
            judge = self.table_col[row][token]
            if judge != -1: # 分析表匹配成功
                self.signRpush.push(self.SignStack.pop())
                self.tokenRpush.push(['','back',''])
```

```
                rig = self.grammar[judge]['right']
                length = len(rig)
                self.signRpop.push(length)
                for i in range(length):
                    if rig[length - 1 - i] != 'NULL':
                        self.SignStack.push(rig[length - 1 - i])
                # 调用语法树搭建程序
                PreNode = predict1(judge + 1, syntax_tree, toke, PreNode)
            else:
                # 分析表匹配失败，调用处理语法错误检测程序
                errJudge, ErrImag = self.dealError.run(self.SignStack, self.TokenStack,
self.signRpush, self.signRpop, self.tokenRpush)
                Err = {'line': 0, 'message': ' '}
                Err['line'] = int(toke[0])
                Err['message'] = ErrImag
                self.errImag.append(Err)
                if not errJudge:
                    break
        else:
            if sign == token:  # 相等则进行匹配
                self.signRpush.push(self.SignStack.pop())
                self.signRpop.push(0)
                self.tokenRpush.push(self.TokenStack.pop())
            else:  # 不相等出错，调用处理语法错误检测程序
                errJudge, ErrImag = self.dealError.run(self.SignStack, self.TokenStack,
self.signRpush, self.signRpop, self.tokenRpush)
                Err = {'line': 0, 'message': ' '}
                Err['line'] = int(toke[0])
                Err['message'] = ErrImag
                self.errImag.append(Err)
                if not errJudge:
                    break
    if self.TokenStack.peek()[2] != 'EOF':
        if len(self.errImag) == 0:
            Err = {'line': 0, 'message': ' '}
            Err['line'] = int(self.TokenStack.peek()[0])
            Err['message'] = '符号栈仍有残余'
            self.errImag.append(Err)
    else:
        self.runJudge = True
    syntax_tree.getInfNode(self.TreePath)
    self.syntax_tree = syntax_tree
```

递归下降核心代码：

```
int main(){
    input.open("../data/token.txt");
    if(!input) {
        cout<<"Error:cannot find or open the specified file!";
        return -1;
```

```
    }
    output.open("../data/syntax_tree.txt");
    if(!output) {
        cout<<"Error:cannot find or open the specified file!";
        return -1;
    }

    Node *head=parse();
    print_tree(head,0);
    if(flag) return -1;
    return 0;
}

Node* parse(){
    read_token();
    Node *t=program();
    if(token!="EOF")
        error(line,"bad end");
    return t;
}

Node* program(){
    Node *t=program_head();
    Node *q=declare_part();
    Node *s=program_body();
    Node *root=init_node();

    root->nodekind=ProK;
    root->child[0]=t;
    root->child[1]=q;
    root->child[2]=s;

    if(token!=".")
        error(line,"there id no . in the end");
    read_token();
    return root;
}

Node *program_head(){
    Node *t=init_node();
    t->nodekind=PheadK;

    if(token!="PROGRAM")
        error(line,"no correct program_head");

    read_token();
    if(type=="ID")
        t->name[0]=token;
    else
```

```
        error(line,"no correct program_head");
    read_token();
    return t;
}

Node *declare_part(){
    Node *type_t=init_node();
    type_t->nodekind=TypeK;
    type_t->child[0]=TypeDec();

    Node *var_t=init_node();
    var_t->nodekind=VarK;
    var_t->child[0]=VarDec();

    Node *proc_deck_t=init_node();
    proc_deck_t->nodekind=ProcDecK;
    proc_deck_t->child[0]=ProcDec();
    type_t->sibling=var_t;
    var_t->sibling=proc_deck_t;

    return type_t;
}

Node *program_body(){
    Node *t=init_node();
    t->nodekind=StmLK;
    if(token=="BEGIN"){
        read_token();
        t->child[0]=StmList();
    }
    else error(line,"there is no BEGIN to match");
    if(token!="END")
        error(line,"there is no END to match");
    read_token();
    return t;
}
```

语义分析:

```
class Node:
    def __init__(self, line, val, deep):
        self.child = []
        self.val = val
        self.deep = deep
        self.line = str(line + 1)
        self.converse(val)

    def __str__(self):
        return str(self.__dict__)
```

```
    def print(self):
        print(str(json.dumps(self.__dict__)))

    def converse(self, val):
        vals = val.split(" ")
        self.nodeKind = vals[0]
        self.rawline = str(int(vals[1]) + 1)
        vals = vals[2:]

        self.kind = ""
        self.idnum = 0  # 一个节点中的标识符的个数
        self.name = []
        self.attr = {}

        # ProK, PheadK, TypeK, VarK, ProDecK, StmLK, DecK, Stmtk, ExpK
        if self.nodeKind == 'DecK':
            if vals[0] == 'valparamType' or vals[0] == "varparamType":
                self.attr['paramt'] = vals[0]
                vals = vals[1:]
            self.kind = vals[0]
            vals = vals[1:]
            if self.kind == "IdK":
                self.realKind = vals[0]
                vals = vals[1:]
            # ArrayK, CharK, IntegerK, RecordK, IdK
            if self.kind == 'ArrayK':
                self.attr['low'] = vals[0]
                self.attr['up'] = vals[1]
                self.attr['childType'] = vals[2]
                vals = vals[3:]
        elif self.nodeKind == 'StmtK':
            # IfK WhileK AssignK ReadK WriteK CallK ReturnK
            if vals[0] != "" or vals[0] != " ":
                self.kind = vals[0]
            vals = vals[1:]
        elif self.nodeKind == 'ExpK':
            # OpK ConstK IdK
            self.kind = vals[0]
            vals = vals[1:]
            if vals[0] in ("IdV", "ArrayMembV", "FieldMembV"):
                self.attr['varkind'] = vals[0]
                vals = vals[1:]
            if self.kind == 'OpK':
                self.attr['op'] = vals[0]
            if self.kind == 'ConstK':
                self.attr['val'] = vals[0]
        for x in vals:
            if x != "":
                self.idnum += 1
```

```
                self.name.append(x)
        # self.type_name = type_name

def generate_node(tree_path):
    level_list = {}
    with open(tree_path) as f:
        lines = f.readlines()
        for i in range(len(lines)):
            line = lines[i].replace("\n", "")
            bn = 0
            j = 0
            for j in range(len(line)):
                if line[j] != " ":
                    break
                else:
                    bn += 1
            line = line[j:]
            level = int(bn / 3)
            node = Node(i, line, level)
            if level not in level_list:
                level_list[str(level)] = [node]
                if level > 0:
                    list = level_list[str(level - 1)]
                    list[len(list) - 1].child.append(node)
    return level_list.get("0")[0]

class DefaultKind:
    def __init__(self, kind):
        self.kind = kind

class Kind:
    def __init__(self, node, body=None):
        self.kind = node.kind
        self.size = 0
        if node.kind == 'ArrayK':
            indexTy = {"low": node.attr["low"], "up": node.attr["up"]}
            elemTy = Kind(DefaultKind(node.attr["childType"]))).__dict__
            self.arrayAttr = {"indexTy": indexTy, "elemTy": elemTy}
            self.size = elemTy["size"] * (int(node.attr["up"]) - int(node.attr["low"]))
            self.arrayKind = elemTy["kind"]
        if node.kind == 'RecordK':
            for x in body:
                self.size += x.size
        if node.kind == 'IntegerK':
            self.size = 2
        if node.kind == 'CharK':
            self.size = 1

    def __str__(self):
```

```
        return str(self.__dict__)

class SymbolTable:
    def __init__(self, node, name, level, off, body=None, params=None, ifType=False):
        self.kind = node.kind
        self.name = name
        self.level = level
        self.off = off
        self.body = None
        self.params = None
        self.ifType = ifType
        if params is not None:
            self.params = params

        if body is not None:
            tmp = []
            for x in body:
                flag = False
                for i in tmp:
                    if x.name[0] == i.name:
                        flag = True
                if flag:
                    error(node.rawline,  f"record  {name}  field  member  {x.name[0]}
duplicated")
                    continue
                y = Kind(x)
                y.name = x.name[0]
                tmp.append(y)

            self.body = tmp
        self.typePtr = Kind(node, self.body)

    def __str__(self):
        s = ""
        if self.body is not None:
            for x in self.body:
                s += str(x.__dict__)
        return     f"kind:{self.kind},     name:{self.name},     level:{self.level},
typePtr:{self.typePtr.__dict__},          body:{s},          params:{self.params},
ifType:{self.ifType}"

def getKind(node):
    if node.kind == "ConstK":
        if str.isdigit(node.name[0]):
            return "IntegerK"
        if re.match(r"\'[a-zA-Z]\'", node.name[0]):
            return "CharK"

    if node.kind == "IdK":
```

```
        kind = node.attr["varkind"]
        v = find(node.name[0])
        if v is None:
            error(node.rawline, "val find failed:", node.name[0])
            return None
        if ck(kind, v.kind) is False:
            error(node.rawline, "val kind errored:", node.name[0], kind, v.kind)
            return None
        if kind == "IdV":
            return v.kind
        if kind == "ArrayMembV":
            if len(node.child) == 1:
                x = node.child[0]
                id = x.name[0]
                l = int(v.typePtr.arrayAttr["indexTy"]["low"])
                r = int(v.typePtr.arrayAttr["indexTy"]["up"])
                if str.isdigit(id) is False:
                    if getKind(x) != "IntegerK":
                        error(node.rawline,  f"array  index  illegal:  {createName(x)},
kind: {getKind(x)}")
                elif int(id) < l or int(id) >= r:
                    error(node.rawline, "array index over range:", f"index:{id}, l:{l},
r:{r}")
            else:
                error(node.rawline, "array cant operate directed:", node.name[0])
            return v.typePtr.arrayKind
        if kind == "FieldMembV":
            nd = None
            for x in v.body:
                if x.name == node.child[0].name[0]:
                    nd = x
            if nd is None:
                error(node.rawline,  f"record  {node.name[0]}  not  have  the  member
{node.child[0].name[0]}")
                return None
            if ck(node.child[0].attr["varkind"], nd.kind) is False:
                error(node.rawline,        f"record        {node.name[0]}        member
{node.child[0].name[0]} kind err: {nd.kind}, ",
                      node.child[0].attr["varkind"])
                return None
            for x in node.child:
                for y in x.child:
                    generate_table(y)
            return getFieldKind(nd)
    if node.kind == 'OpK':
        return operator(node, node.name[0])

def operator(node, op):
    kindList = []
```

```
    for x in node.child:
        kindList.append(generate_table(x))
    if len(kindList) == 0:
        error(node.rawline, "operate not have child")
        return None
    for i in range(len(kindList)):
        if kindList[i] is None:
            return None
        elif kindList[i] not in ("IntegerK", "CharK"):
            error(node.rawline, op, "illegal operate kind:", kindList[i])
        elif kindList[i] != kindList[0]:
            error(node.rawline, op, "operate failed:",
                  [(node.child[x].name[0], kindList[x]) for x in range(len(kindList))])
            return None
        elif op in ("+", "-", "*", "/") and kindList[i] == "CharK":
            error(node.rawline, op, "can't sub char",
                  [(node.child[x].name[0], kindList[x]) for x in range(len(kindList))])
            return None

    return kindList[0]

def generate_table(node):
    global sl, scope,off
    # ProK, PheadK, TypeK, VarK, ProDecK, StmLK, DecK, Stmtk, ExpK
    if node.nodeKind == "DecK":
        for x in node.name:
            if find(x, exist=True) is not None:
                error(node.rawline, "val duplicated:", x)
                continue

            body = None
            if node.kind == "RecordK":
                body = []
                for y in node.child:
                    body.append(y)


            tab = CallSymbolTable(node, x, level=sl, off=off, body=body)
            if tab is None:
                continue

            if len(scope[sl]) == 0:
                tab.off = 0
            else:
                tmp = scope[sl][-1]
                tab.off = tmp.typePtr.size + tmp.off

            scope[sl].append(tab)
            all_scope[sl].append(tab)
```

```
        if node.kind == "RecordK":
            return
        for x in node.child:
            generate_table(x)
    elif node.nodeKind == "ProcDecK" and node.idnum > 0:
        if find(node.name[0], exist=True) is not None:
            error(node.rawline, "val duplicated:", node.name[0])
            return
        params = []
        for x in node.child:
            if x.nodeKind == "DecK":
                for y in x.name:
                    if y != " " and y != "":
                        params.append({"kind": x.kind, "name": y})
        node.kind = "ProcDecK"
        tab = CallSymbolTable(node, node.name[0], level=sl, off=off, params=params)
        if tab is None:
            return

        if len(scope[sl]) == 0:
            tab.off = 0
        else:
            tmp = scope[sl][-1]
            tab.off = tmp.typePtr.size + tmp.off

        scope[sl].append(tab)
        all_scope[sl].append(tab)

        sl += 1
        scope.append([])
        all_scope.append([])
        for x in node.child:
            generate_table(x)
        sl -= 1
        scope = scope[:-1]
    elif node.nodeKind == "StmtK":
        # IfK WhileK AssignK ReadK WriteK CallK ReturnK
        # print("kind:", node.kind)
        if node.kind == "CallK":
            pro = find(node.name[0])
            if pro is None:
                error(node.rawline, "procDeck find failed:", node.name[0])
                return
            elif pro.kind != "ProcDecK":
                error(node.rawline, "procDeck kind error:", node.name[0], pro.kind)
                return
            params = []
            for x in node.child:
                if x.kind == "OpK":
```

```
                kind = operator(x, x.name[0])
                if kind is None:
                    return
            else:
                kind = getKind(x)
                if kind is None:
                    error(x.rawline, "val find failed:", x.name[0])
                    return
            params.append(kind)
        proParams = [x["kind"] for x in pro.params]
        # print(params, pro.params)
        if len(params) != len(proParams):
            error(node.rawline, "call failed:", params, proParams)
            return
        for i in range(len(params)):
            if params[i] != proParams[i]:
                error(node.rawline, "call failed:", params, proParams)
                return
        return
    if node.kind == "IfK":
        for x in node.child:
            generate_table(x)
    if node.kind == "AssignK":
        if node.child[0].kind != "IdK":
            error(node.rawline, "AssignK left kind illegal", node.name[0])
        return operator(node, "=")
    if node.kind == "ReadK":
        if find(node.name[0]) is None:
            error(node.rawline, "val find failed:", node.name[0])
        return
    if node.kind == "WriteK":
        return operator(node, "write")
    if node.kind == "ReturnK":
        return
    if node.kind == "WhileK":
        for x in node.child:
            generate_table(x)
        return
elif node.nodeKind == "ExpK":
    # OpK ConstK IdK
    if node.kind == "OpK":
        return operator(node, node.name[0])

    if node.kind in ("IdK", "ConstK"):
        return getKind(node)
elif node.nodeKind == "TypeK":
    for x in node.child:
        if x.kind == "RecordK":
            generate_table(x)
```

```
            continue
        if find(x.name[0], exist=True) is not None:
            error(node.rawline, "type duplicated:", x.name[0])
            continue
        tab = CallSymbolTable(x, x.name[0], level=sl, off=off, ifType=True)
        if tab is None:
            continue

        if len(scope[sl]) == 0:
            tab.off = 0
        else:
            tmp = scope[sl][-1]
            tab.off = tmp.typePtr.size + tmp.off

        scope[sl].append(tab)
        all_scope[sl].append(tab)
    else:
        for x in node.child:
            generate_table(x)
    return
```