2020 编译技术实验课设计文档

78066014 邓奇恩

一、词法分析设计

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
标识符	IDENFR	else	ELSETK	-	MINU	=	ASSIGN
整形常量	INTCON	switch	SWITCHTK	*	MULT	;	SEMICN
字符常量	CHARCON	case	CASETK	/	DIV	,	COMMA
字符串	STRCON	default	DEFAULTTK	<	LSS	(	LPARENT
const	CONSTTK	while	WHILETK	<=	LEQ	)	RPARENT
int	INTTK	for	FORTK	>	GRE	[	LBRACK
char	CHARTK	scanf	SCANFTK	>=	GEQ	]	RBRACK
void	VOIDTK	printf	PRINTFTK	==	EQL	{	LBRACE
main	MAINTK	return	RETURNTK	!=	NEQ	}	RBRACE
if	IFTK	+	PLUS	:	COLON		

下图为词法分析程序的算法框图：

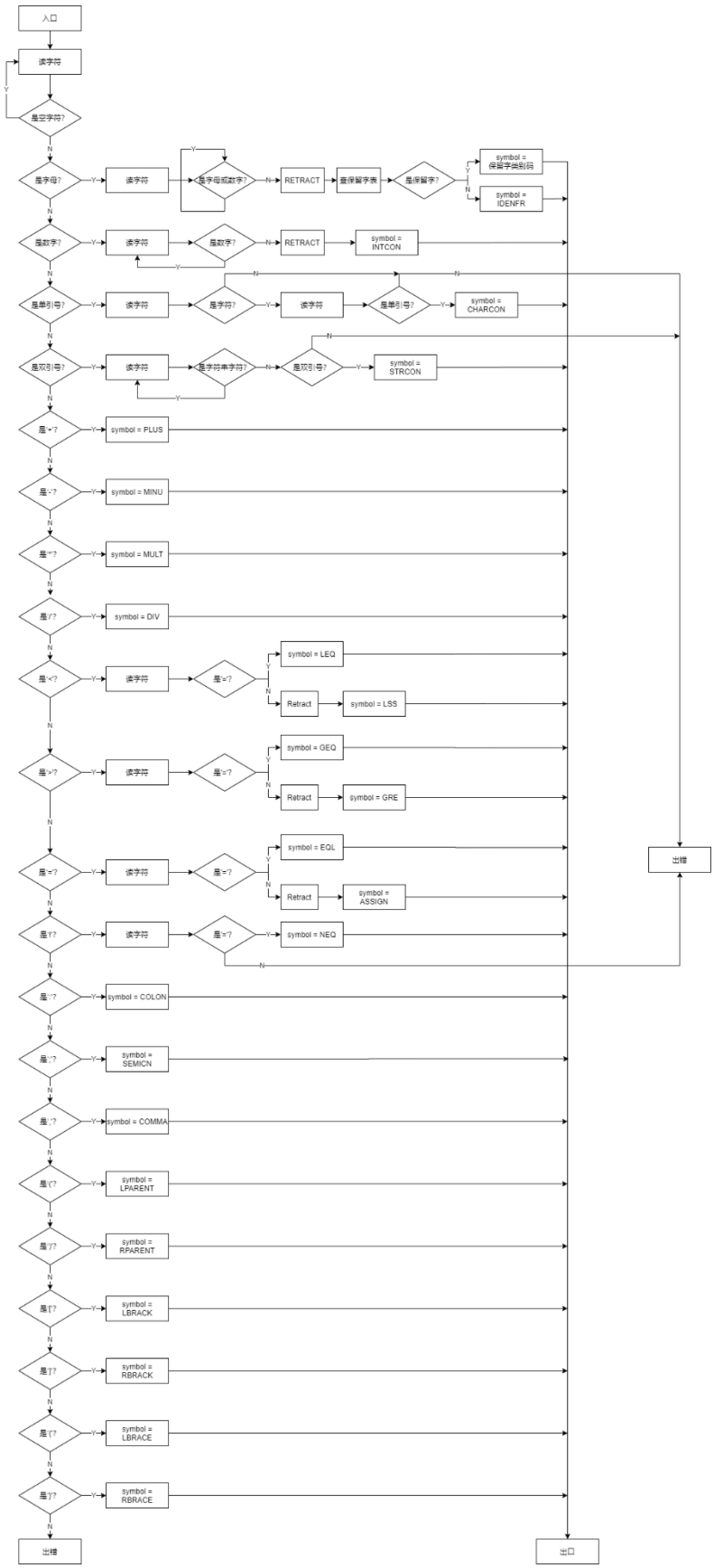
其中，**空字符**是 ‘ ’, ‘\n’, ‘\t’, ‘\v’, ‘\f’, ‘\r’

单引号内的字符是 ‘+’, ‘-’, ‘*’, ‘/’, ‘<’, ‘_’, ‘a’, … ‘z’, ‘A’, …, ‘Z’, ‘0’, …, ‘9’

字符串字符是十进制编码为 32,33,35-126 的 ASCII 字符。

Retract 是把读入的最后一个字符回退。

编码之前的设计没有包含错误处理的情况。编码之后，对一些错误情况进行了处理。


入口
读字符
是空字符?
是字母?
读字符
是字母或数字?
RETRACT
查保留字表
是保留字?
symbol = 保留字类别码
symbol = IDENFR
是数字?
读字符
是数字?
RETRACT
symbol = INTCON
是单引号?
读字符
是字符?
读字符
是单引号?
symbol = CHARCON
是双引号?
读字符
是字符串字符?
是双引号?
symbol = STRCON
是+?
symbol = PLUS
是-?
symbol = MINU
是*?
symbol = MULT
是/?
symbol = DIV
是<?
读字符
是=?
symbol = LEQ
Retract
symbol = LSS
是>?
读字符
是=?
symbol = GEQ
Retract
symbol = GRE
是=?
读字符
是=?
symbol = EQL
Retract
symbol = ASSIGN
是!?
读字符
是=?
symbol = NEQ
出错
是:?
symbol = COLON
是;?
symbol = SEMICN
是,?
symbol = COMMA
是(?
symbol = LPARENT
是)?
symbol = RPARENT
是[?
symbol = LBRACK
是]?
symbol = RBRACK
是{?
symbol = LBRACE
是}?
symbol = RBRACE
出错
出口

二、语法分析阶段设计文档

语法分析子程序定义了一个SyntaxAnalyzer Class。

```
class SyntaxAnalyzer {
public:
	SyntaxAnalyzer(vector<Token>& Tokens);
	vector<string> outputVec;

private:
	int idx;
	vector<Token> tokenVec;
	string symbol;
	string token;
	unordered_map<string, string> functionMap;

	void insertToOutputVec(const char* str);
	void addFunctionToMap(string functionName, string returnType);
	string findFunctionFromMap(string functionName);
	void nextSymbol();
	void retractSymbol();
	void charString();   //<字符串>
	void program();      //<程序>
	void constDeclare(); //<常量说明>
	void constDefine(); //<常量定义>
	void unsignedInteger(); //<无符号整数>
	void integer(); //<整数>
	void declareHead(); //<声明头部>
	void constant(); //<常量>
	void varDeclare(); //<变量说明>
	void varDefine();    //<变量定义>
	void varDefineNoInit(); //<变量定义无初始化>
	void varDefineInit(); //<变量定义及初始化>
	void functionReturnDefine(); //<有返回值函数定义>
	void functionNoReturnDefine(); //<无返回值函数定义>
	void compoundStatement(); //<复合语句>
	void parameterTable();      //<参数表>
	void mainFunction(); //<主函数>
	void expression(); //<表达式>
	void term(); //<项>
	void factor(); //<因子>
	void statement(); //<语句>
	void assignStatement(); //<赋值语句>
	void conditionalStatement(); //<条件语句>
	void condition(); //<条件>
	void loopStatement(); //<循环语句>
	void stepLength(); //<步长>
	void caseStatement(); //<情况语句>
	void caseTable(); //<情况表>
	void caseChildStatement();  //<情况子语句>
	void defaultChildStatement(); //<缺省>
	//void callFunctionReturn(); //<有返回值函数调用语句>   moved to callFunction
	//void callFunctionNoReturn(); //<无返回值函数调用语句>   moved to callFunction
	void callFunction(); //<有返回值函数调用语句> and <无返回值函数调用语句>
	void paramValueTable(); //<值参数表>
	void statementList(); //<语句列>
```

```
        void readStatement(); //<读语句>
        void writeStatement(); //<写语句>
        void returnStatement(); //<返回语句>
};
```

对SyntaxAnalyzer Class调用构造方法SyntaxAnalyzer(vector<Token>& Tokens) 可以在SyntaxAnalyzer Class中生成一个vector<string>类型的outputVec，outputVec中存储着语法分析程序的输出语句。Tokens是词法分析子程序解析出来的所有Token。Token类中存储着词法分析阶段解析出的token和其对应的symbol。Token类定义如下：

```
class Token {
public:
        Token();
        Token(string token, string symbol);
        string getToken();
        string getSymbol();
        void setToken(string token);
        void setSymbol(string symbol);

private:
        string token;
        string symbol;
};
```

SyntaxAnalyzer Class 中几个较重要的变量如下。

```
        int idx;      //当前访问的tokenVec的idx
        vector<Token> tokenVec;     //词法分析解析出的Token（Token类包含token和symbol）
        string symbol;       //当前在tokenVec访问的Token的symbol
        string token;        //当前在tokenVec访问的Token的token
        unordered_map<string, string> functionMap;       //函数定义时，把函数名存储为key，若函数是有返回值函数，则value存储为“RETURNFUNCTION”，若函数为无返回值函数，则value存储为“VOIDFUNCTION”，若函数未定义，返回“FunctionNotFound”。
```

nextSymbol()和retractSymbol()是语法分析程序中最常用的函数，他们的作用分别是从tokenVec中得到下一个Token，以及把Token回退为上一个Token。这两个函数的实现方式如下。

```
void SyntaxAnalyzer::nextSymbol() {
        idx++;
        if (idx < tokenVec.size()) {
                symbol = tokenVec[idx].getSymbol();
                token = tokenVec[idx].getToken();
                outputVec.push_back(symbol + " " + token);
        }
        else {
                Error("no more symbol");
        }
}
void SyntaxAnalyzer::retractSymbol() {
        idx--;
        symbol = tokenVec[idx].getSymbol();
        token = tokenVec[idx].getToken();
        outputVec.pop_back();
}
```

语法分析程序的实现方式大致上与文法定义一样直接明了，只是有些文法的右方非常相似，是可以提取出来的，这些我在编码前没有考虑清楚，导致编码当中需要进行修改，把文法的右方提取出来。

比如说，语句可以推出＜有返回值函数调用语句＞ 或＜无返回值函数调用语句＞，而两种语句的定义非常相似，所以我把对这两种语句的判断放在同一个函数 callFunction 里。

＜语句＞ ::= ＜循环语句＞ | ＜条件语句＞| ＜有返回值函数调用语句＞; |＜无返回值函数调用语句＞; | ＜赋值语句＞; | ＜读语句＞; | ＜写语句＞; | ＜情况语句＞ | ＜空＞;|＜返回语句＞; | '{'＜语句列＞'}'

```
void SyntaxAnalyzer::callFunction() {
	if (symbol == "IDENFR") {
		string functionName = token;
		string retStr = findFunctionFromMap(functionName);
		if (retStr == "FunctionNotFound") {
			Error("calling an undefined function");
		}
		else {
			nextSymbol();
			if (symbol == "LPARENT") {
				nextSymbol();
				if (symbol == "RPARENT") {  //参数为空
					retractSymbol();
					insertToOutputVec("<值参数表>");
					nextSymbol();
				}
				else {
					paramValueTable();
					nextSymbol();
				}
				if (symbol == "RPARENT") {
					if (retStr == "RETURNFUNCTION") {
						insertToOutputVec("<有返回值函数调用语句>");
					}
					else if (retStr == "VOIDFUNCTION") {
						insertToOutputVec("<无返回值函数调用语句>");
					}
				}
				else {
					Error("missing RPARENT");
				}

			}
		}
	}
}
```

三、错误处理阶段设计文档

对于错误 a，在词法分析阶段 lexical analyzer 程序中进行判断。若字符或字符串中出现非法符号、或出现空字符、空字符串，则报错误 a。

其余的错误由语法分析程序处理。语法分析程序增加了构建符号表功能。

符号表中每个表项有 name，kind，type，level 这 4 个属性。

name 填入函数、变量、常量名的小写名。

kind 填入选项有 VAR、CONSTTK、RETURNFUNCTION、VOIDFUNCTION。

type 填入选项有 INTTK、CHARTK、VOIDTK、ARRAY。

level 可填入 0 或 1。

若表项为有参数的函数，则使用 symbolTable.setFuncParam(string name, vector<string> param)函数把<name, param>pair 放入 funcParamMap 中。

若表项为有维度的数组，则使用 symbolTable.setArrayAttr(string type, vector<int> dimVec)函数把< <name,level>, <type, dimVec> >放入 arrayAttrMap 中。

语法分析程序采用回溯方法，其递归下降子程序中如下。语法分析程序函数入口为 SyntaxAnalyzer。

其中，

nextSymbol：获取下一个 symbol。

retractSymbol：回退为上一个 symbol。

CHECKRPARENT：若当前 symbol 不是 RPARENT，则报错误 l，并调用 retractSymbol。

CHECKBRACK：若当前 symbol 不是 RBRACK，则报错误 m，并调用 retractSymbol。

CHECKSEMICN：若当前 symbol 不是 SEMICN，则报错误 k，并调用 retractSymbol。

skip()：不断调用 nextSymbol 直到当前符号为括号中的符号。如 skip（SEMICN，RPARENT）会跳到下一个最靠近的 SEMICN 或 RPARENT 的位置。

getType：从符号表查找当前 IDENFR 的 type，若 IDENFR 未定义，则报错误 c。

getKind：从符号表查找当前 IDENFR 的 kind，若 IDENFR 未定义，则报错误 c。

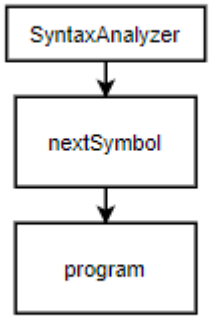
error（a）：报错误 a。

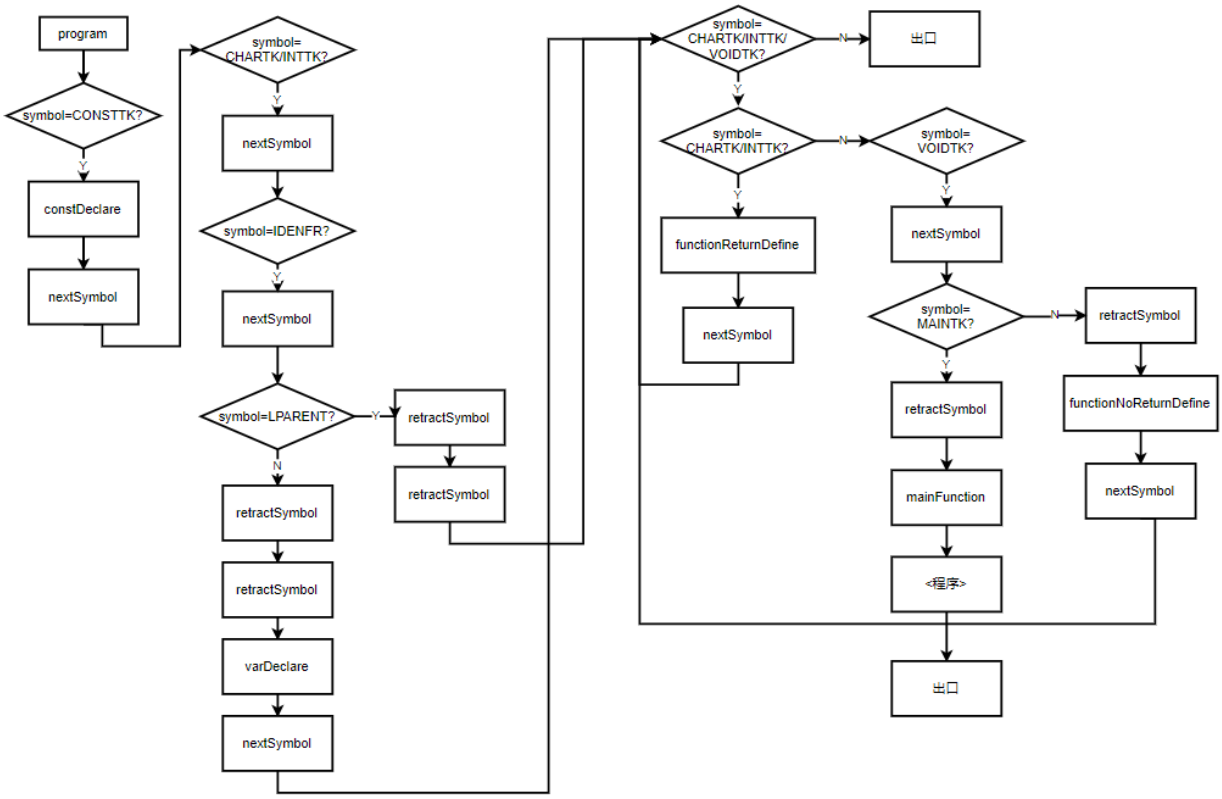
checkDupIdenfr(string name, int level)：在符号表中查找当前 IDENFR 在当前层次是否有重定义，若重定义了就报错误 b，并返回 false。否则返回 true。

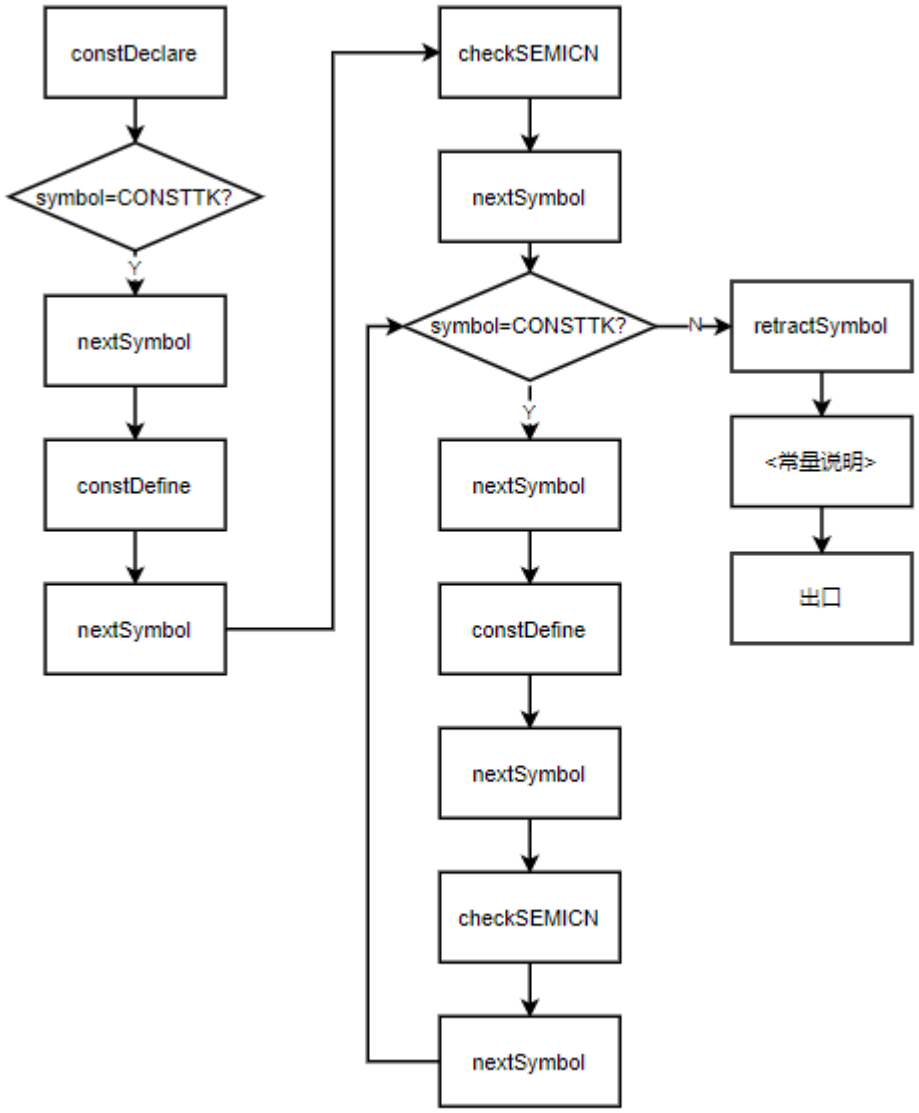
checkIdenfrHasDefined(string name)：在符号表中查找当前 IDENFR 是否已定义，若未定义就报错误 c，并返回 false。否则返回 true。

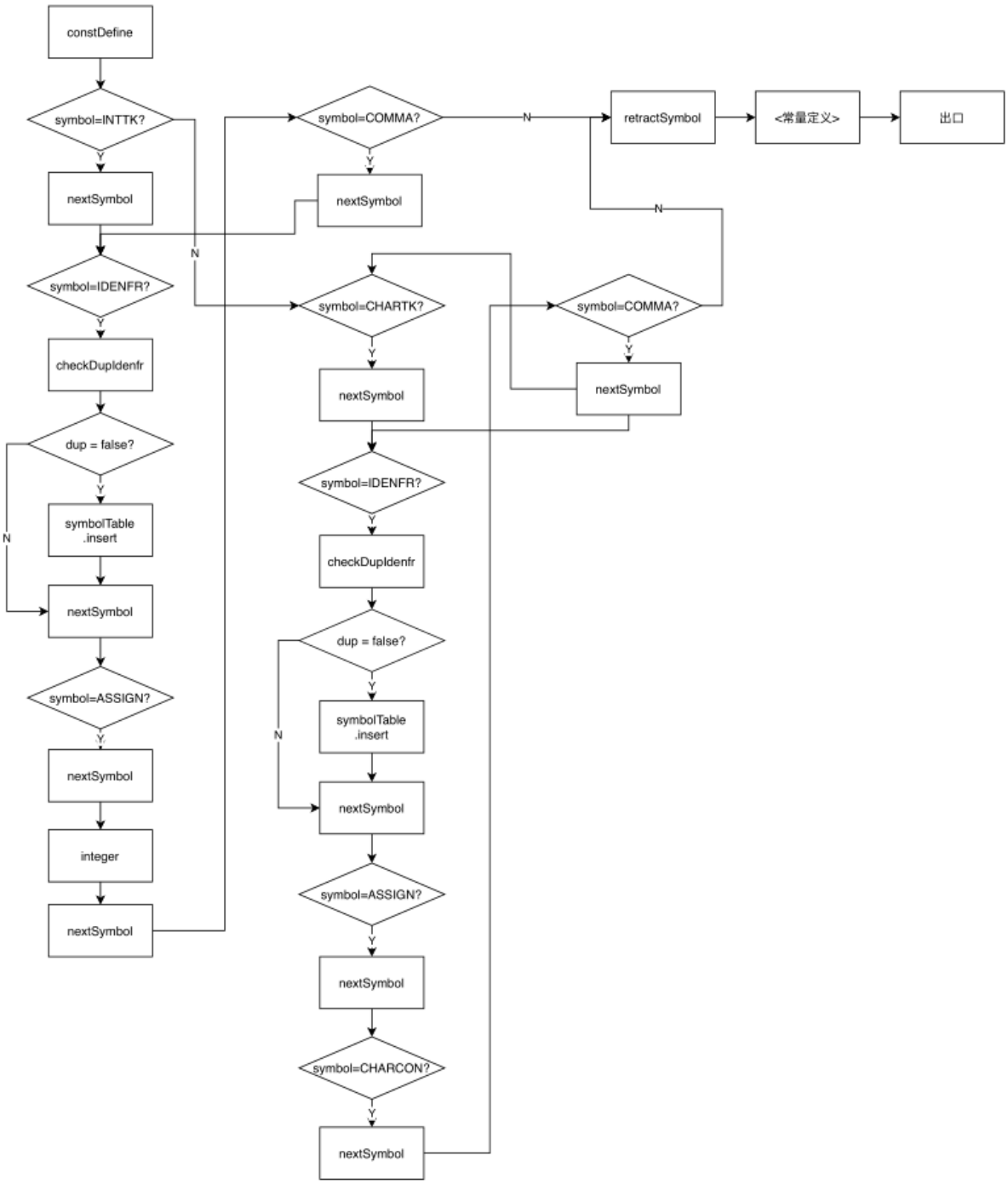
这是语法分析与错误处理程序的递归下降子程序。

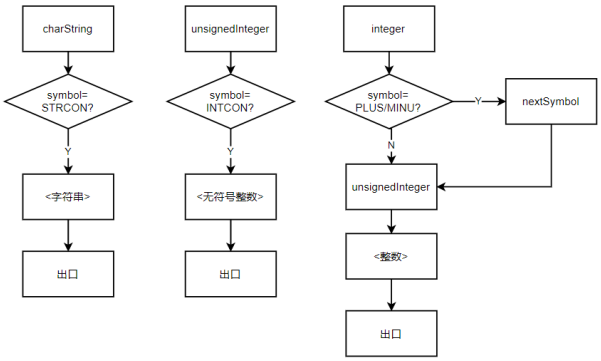
```
void charString();   //<字符串>
void program();      //<程序>
void constDeclare(); //<常量说明>
void constDefine(); //<常量定义>
int unsignedInteger(); //<无符号整数>
void integer(); //<整数>
//string declareHead(); //<声明头部> moved to functionReturnDefine()
void constant(string type); //<常量>
void varDeclare(); //<变量说明>
void varDefine();    //<变量定义>
void varDefineNoInit(string type); //<变量定义无初始化>
void varDefineInit(); //<变量定义及初始化>
void functionReturnDefine(); //<有返回值函数定义>
void functionNoReturnDefine(); //<无返回值函数定义>
void compoundStatement(); //<复合语句>
void parameterTable(string funcName, bool dup); //<参数表>
void mainFunction(); //<主函数>
string expression(); //<表达式>
string term(); //<项>
string factor(); //<因子>
void statement(); //<语句>
void assignStatement(); //<赋值语句>
void conditionalStatement(); //<条件语句>
void condition(); //<条件>
void loopStatement(); //<循环语句>
void stepLength(); //<步长>
void caseStatement(); //<情况语句>
void caseTable(string type); //<情况表>
void caseChildStatement(string type);  //<情况子语句>
void defaultChildStatement(); //<缺省>
//void callFunctionReturn(); //<有返回值函数调用语句>   moved to callFunction
//void callFunctionNoReturn(); //<无返回值函数调用语句>   moved to callFunction
string callFunction(); //<有返回值函数调用语句> and <无返回值函数调用语句>
void paramValueTable(vector<string> param); //<值参数表>
void statementList(); //<语句列>
void readStatement(); //<读语句>
void writeStatement(); //<写语句>
void returnStatement(); //<返回语句>
```

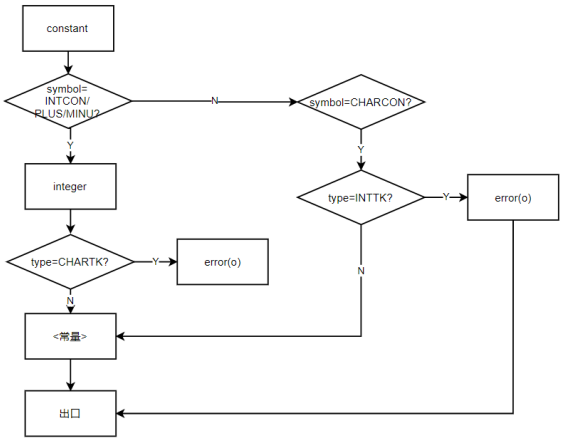


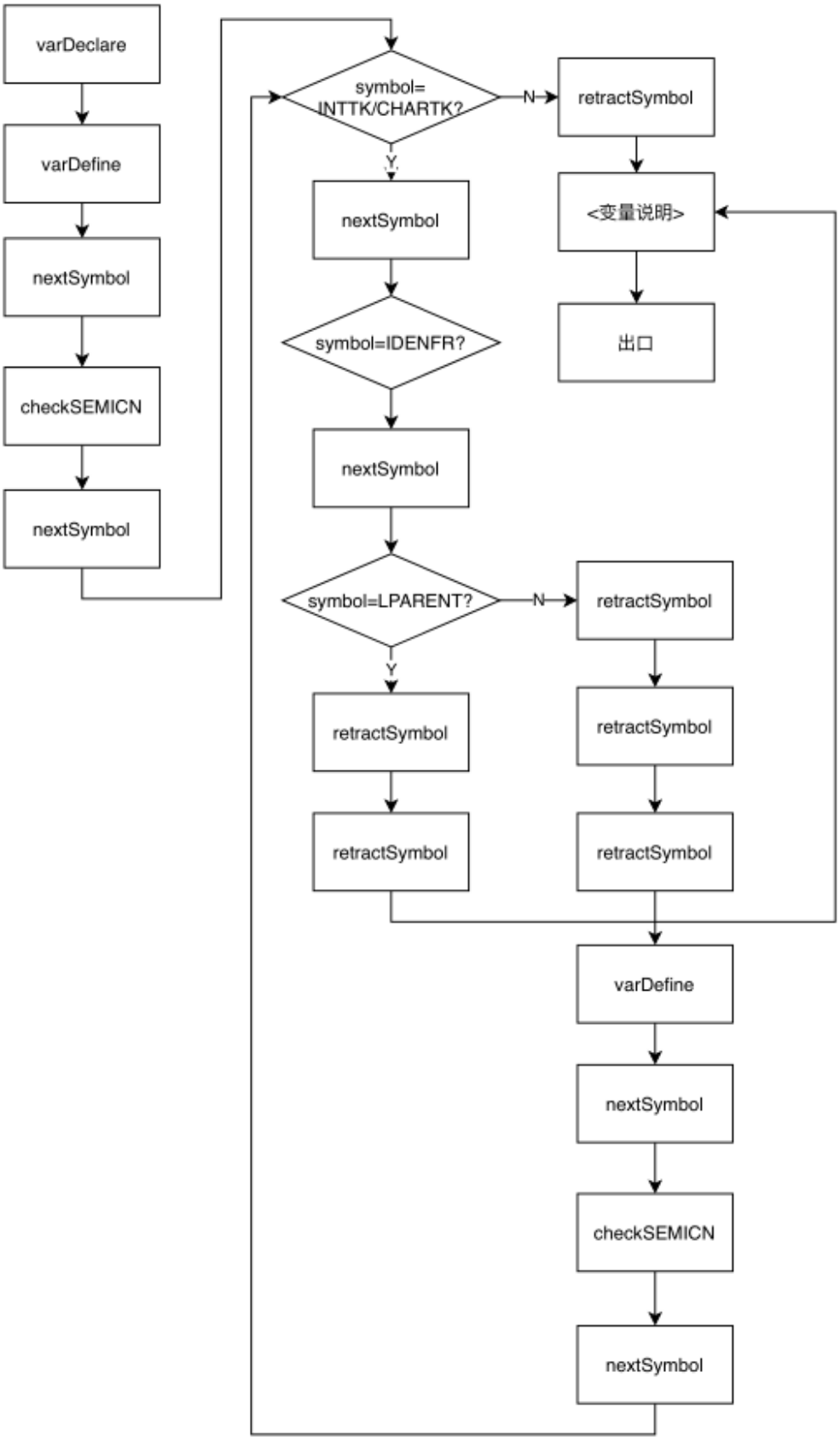

program
symbol=CONSTTK?
Y
constDeclare
nextSymbol
symbol=
CHARTK/INTTK?
Y
nextSymbol
symbol=IDENFR?
Y
nextSymbol
symbol=LPARENT?
Y
retractSymbol
retractSymbol
N
retractSymbol
retractSymbol
varDeclare
nextSymbol
symbol=
CHARTK/INTTK/
VOIDTK?
N
出口
Y
symbol=
CHARTK/INTTK?
N
symbol=
VOIDTK?
Y
functionReturnDefine
nextSymbol
Y
nextSymbol
symbol=
MAINTK?
N
retractSymbol
Y
retractSymbol
functionNoReturnDefine
mainFunction
nextSymbol
<程序>
出口

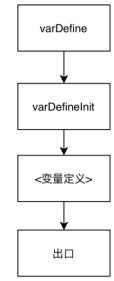

constDeclare
symbol=CONSTTK?
Y
nextSymbol
constDefine
nextSymbol
checkSEMICN
nextSymbol
symbol=CONSTTK?
N
retractSymbol
Y
nextSymbol
constDefine
nextSymbol
checkSEMICN
nextSymbol
<常量说明>
出口

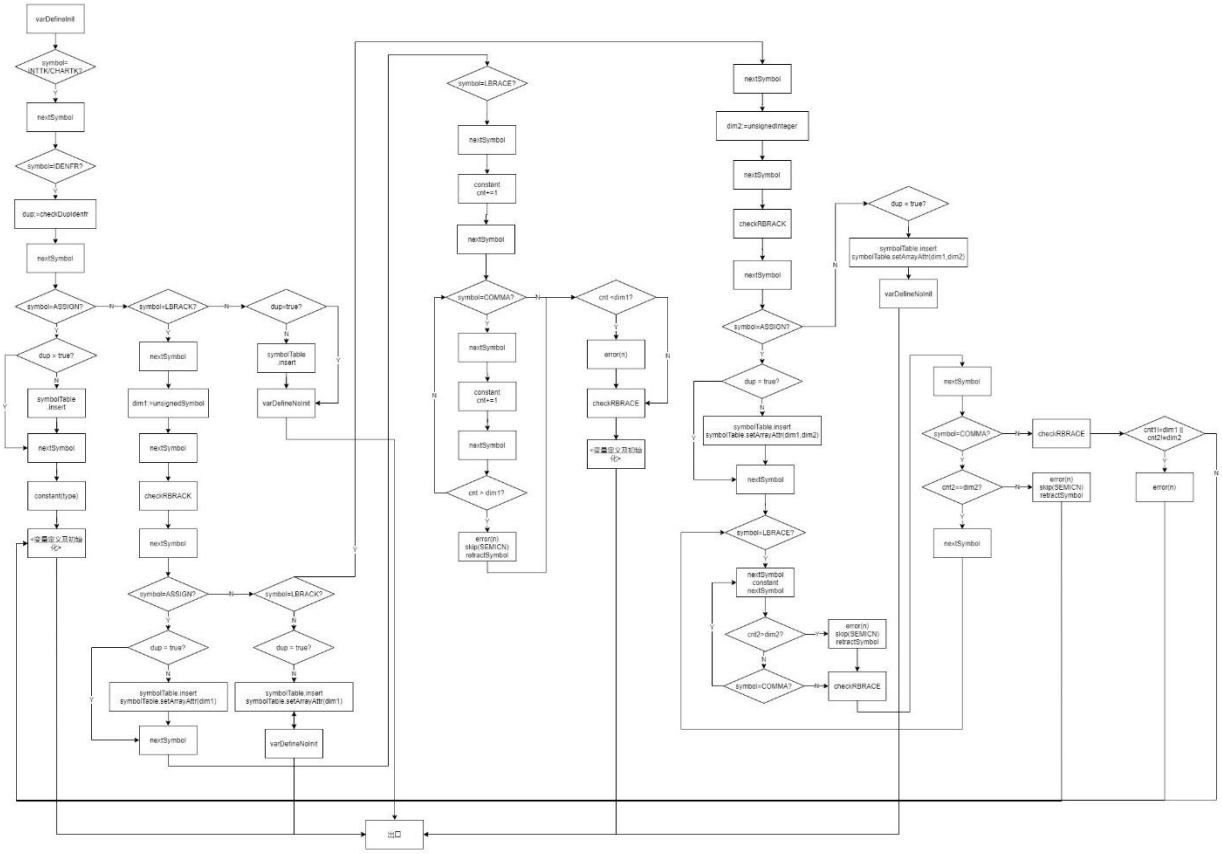

constDefine
symbol=INTTK?
Y
nextSymbol
symbol=IDENFR?
Y
checkDupIdenfr
dup = false?
Y
symbolTable
.insert
N
nextSymbol
symbol=ASSIGN?
Y
nextSymbol
integer
nextSymbol
N
symbol=COMMA?
Y
nextSymbol
N
symbol=CHARTK?
Y
nextSymbol
symbol=IDENFR?
Y
checkDupIdenfr
dup = false?
Y
symbolTable
.insert
N
nextSymbol
symbol=ASSIGN?
Y
nextSymbol
symbol=CHARCON?
Y
nextSymbol
symbol=COMMA?
Y
nextSymbol
N
retractSymbol
<常量定义>
出口

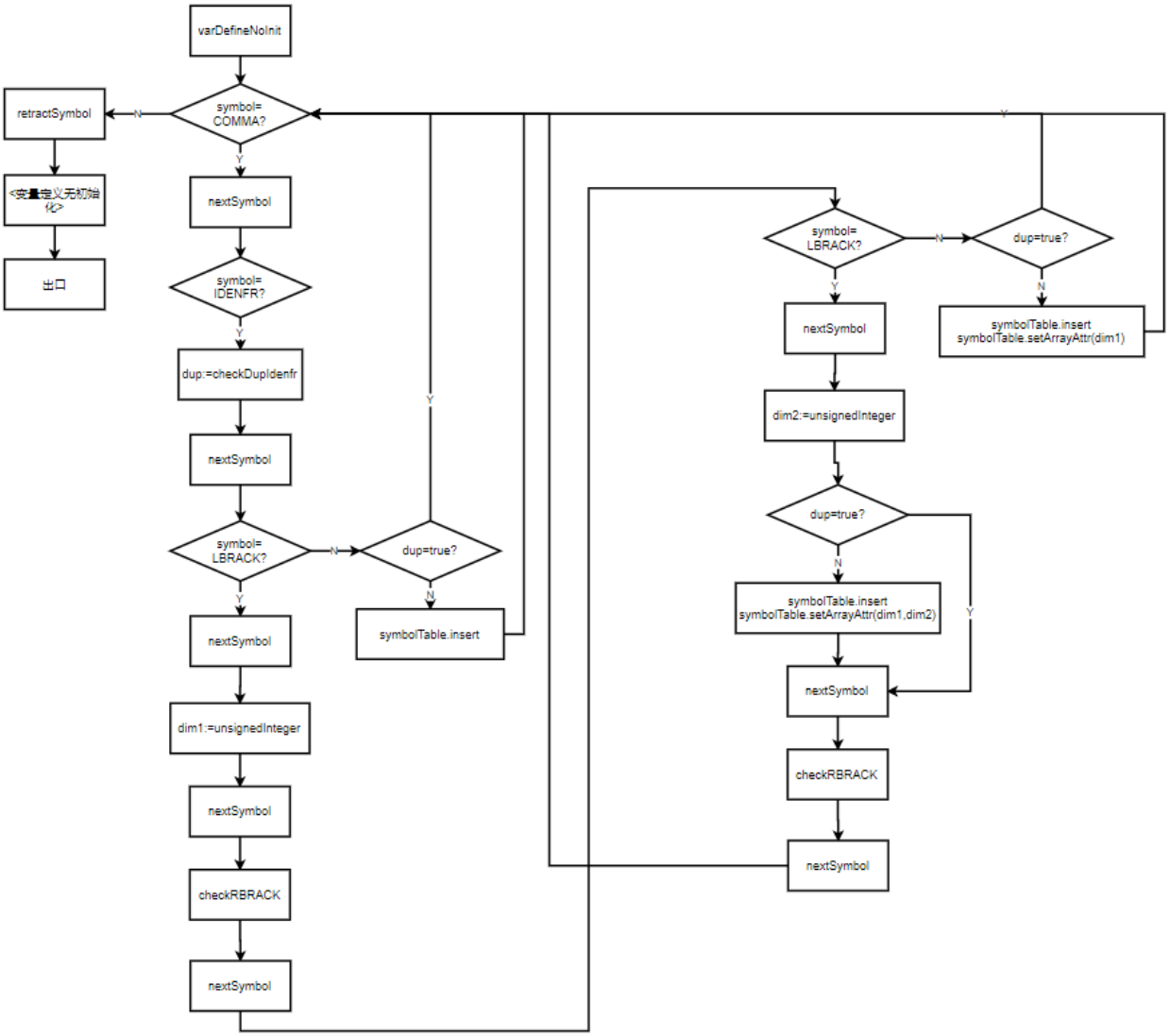

charString
symbol=
STRCON?
Y
<字符串>
出口
unsignedInteger
symbol=
INTCON?
Y
<无符号整数>
出口
integer
symbol=
PLUS/MINU?
Y
nextSymbol
N
unsignedInteger
<整数>
出口

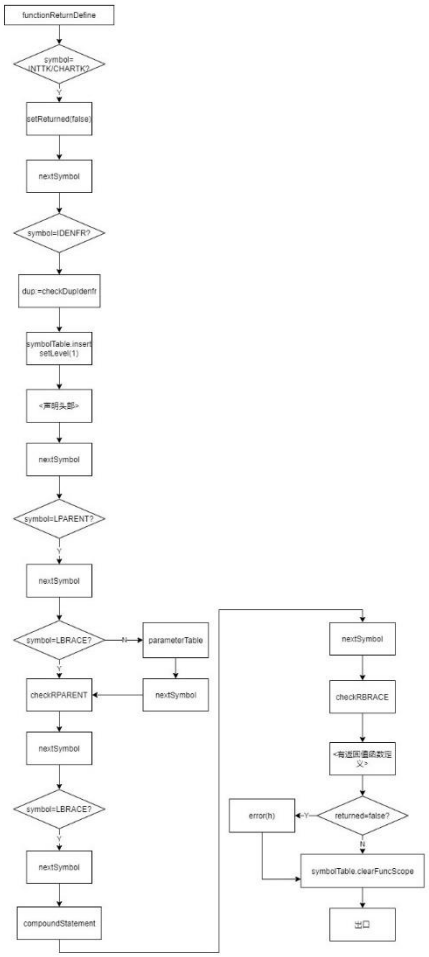

constant
symbol=
INTCON/
PLUS/MINU?
N
symbol=CHARCON?
Y
Y
integer
type=INTTK?
Y
error(o)
type=CHARTK?
Y
error(o)
N
N
<常量>
出口

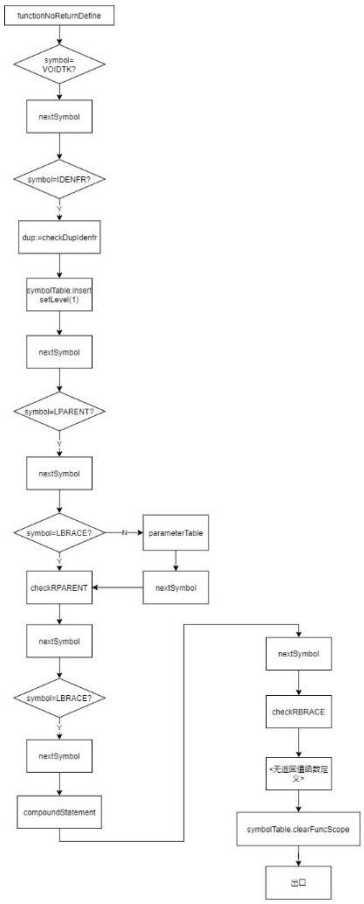

varDeclare
varDefine
nextSymbol
checkSEMICN
nextSymbol
symbol=
INTTK/CHARTK?
N
retractSymbol
<变量说明>
出口
Y
nextSymbol
symbol=IDENFR?
nextSymbol
symbol=LPARENT?
N
retractSymbol
Y
retractSymbol
retractSymbol
retractSymbol
retractSymbol
retractSymbol
varDefine
nextSymbol
checkSEMICN
nextSymbol

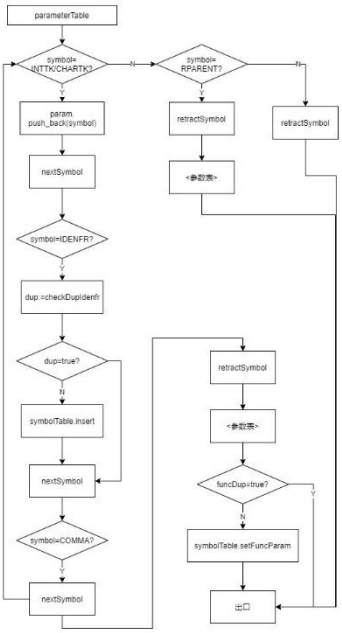

varDefine
varDefineInit
<变量定义>
出口

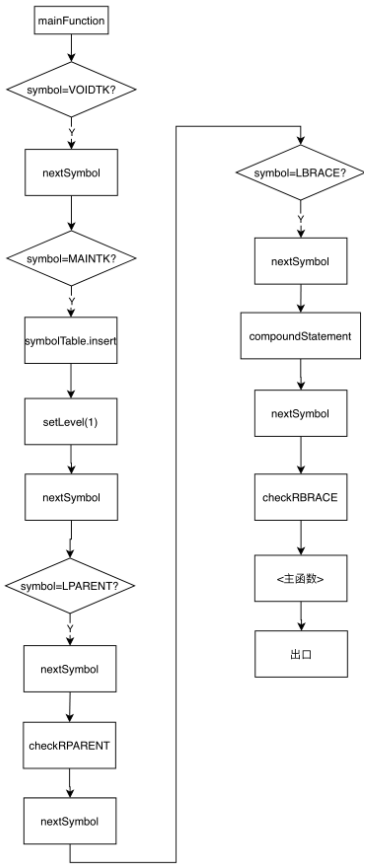

varDefineInit
symbol=INTTK/CHARTK?
nextSymbol
symbol=IDENFR?
dup=checkDupIdenfr
nextSymbol
symbol=ASSIGN?
symbol=LBRACK?
dup=true?
dup = true?
symbolTable insert
nextSymbol
constant(type)
<变量定义及初始化>
nextSymbol
dim1=unsignedSymbol
checkRBRACK
nextSymbol
symbolTable insert
varDefineNoInit
symbol=ASSIGN?
symbol=LBRACK?
dup = true?
dup = true?
symbolTable insert symbolTable.setArrayAttr(dim1)
symbolTable insert symbolTable.setArrayAttr(dim1)
nextSymbol
varDefineNoInit
symbol=LBRACE?
nextSymbol
constant cnt=+1
nextSymbol
symbol=COMMA?
cnt <dim1?
nextSymbol
constant cnt=+1
nextSymbol
cnt > dim1?
error(n) skip(SEMICN) retractSymbol
error(n)
checkRBRACE
<变量定义及初始化>
nextSymbol
dim2=unsignedInteger
nextSymbol
checkRBRACK
nextSymbol
symbol=ASSIGN?
dup = true?
symbolTable insert symbolTable.setArrayAttr(dim1,dim2)
nextSymbol
symbol=LBRACE?
nextSymbol constant nextSymbol
cnt2>dim2?
error(n) skip(SEMICN) retractSymbol
symbol=COMMA?
checkRBRACE
dup = true?
symbolTable insert symbolTable.setArrayAttr(dim1,dim2)
varDefineNoInit
nextSymbol
symbol=COMMA?
checkRBRACE
cnt1<dim1 || cnt2!=dim2
cnt2==dim2?
error(n) skip(SEMICN) retractSymbol
error(n)
nextSymbol
出口

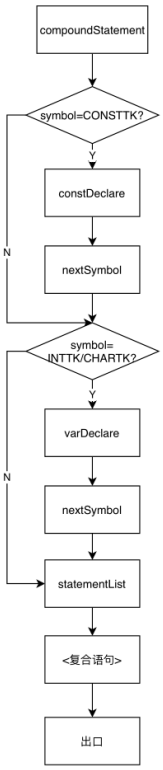

varDefineNoInit
symbol= COMMA?
N
retractSymbol
<变量定义无初始化>
出口
Y
nextSymbol
symbol= IDENFR?
Y
dup:=checkDupIdenfr
nextSymbol
symbol= LBRACK?
N
dup=true?
N
symbolTable.insert
Y
Y
nextSymbol
dim1:=unsignedInteger
nextSymbol
checkRBRACK
nextSymbol
symbol= LBRACK?
N
dup=true?
Y
nextSymbol
dim2:=unsignedInteger
dup=true?
N
symbolTable.insert
symbolTable.setArrayAttr(dim1,dim2)
Y
nextSymbol
checkRBRACK
nextSymbol
Y
N
symbolTable.insert
symbolTable.setArrayAttr(dim1)

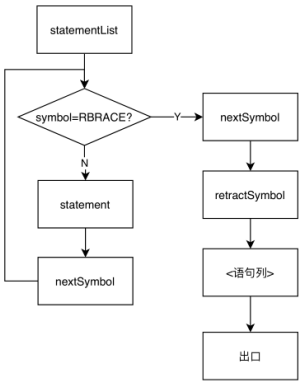

functionReturnDefine
symbol=INTTK/CHARTK?
setReturned(false)
nextSymbol
symbol=IDENFR?
dup:=checkDupIdenfr
symbolTable.insert setLevel(1)
<声明头部>
nextSymbol
symbol=LPARENT?
nextSymbol
symbol=LBRACE?
N
parameterTable
nextSymbol
checkRPARENT
nextSymbol
symbol=LBRACE?
nextSymbol
compoundStatement
nextSymbol
checkRBRACE
<有返回值函数定义>
returned=false?
Y
error(h)
N
symbolTable.clearFuncScope
出口

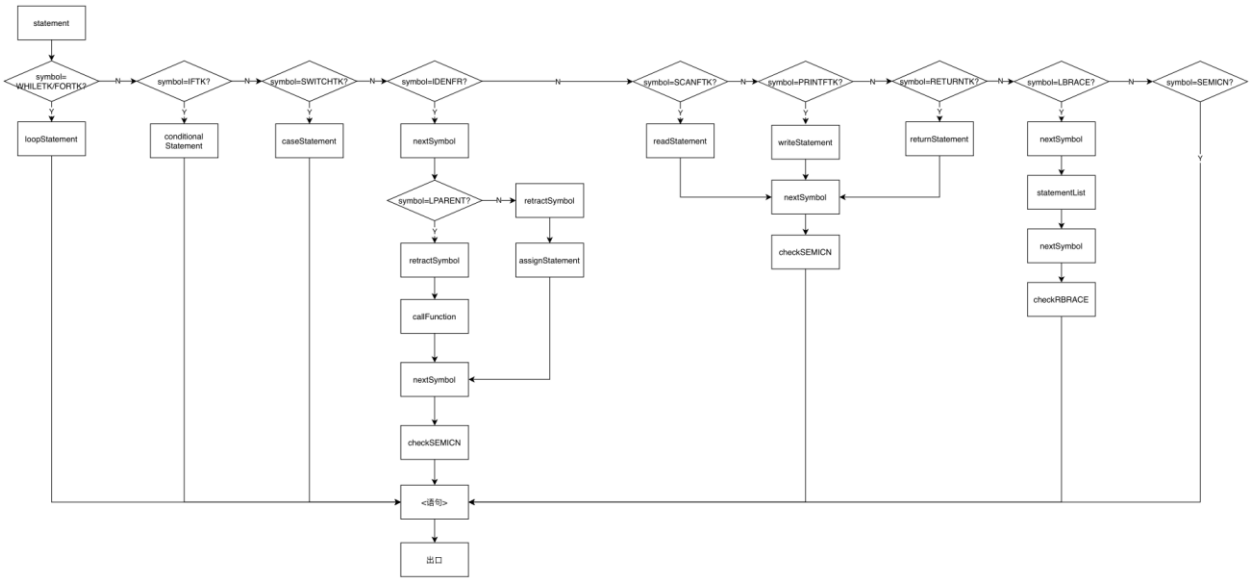

functionNoReturnDefine
symbol=VOIDTK?
nextSymbol
symbol=IDENFR?
dup:=checkDupIdenfr
symbolTable.insert setLevel(1)
nextSymbol
symbol=LPARENT?
nextSymbol
symbol=LBRACE?
N
parameterTable
checkRPARENT
nextSymbol
nextSymbol
symbol=LBRACE?
nextSymbol
compoundStatement
nextSymbol
checkRBRACE
<无返回值函数定义>
symbolTable.clearFuncScope
出口

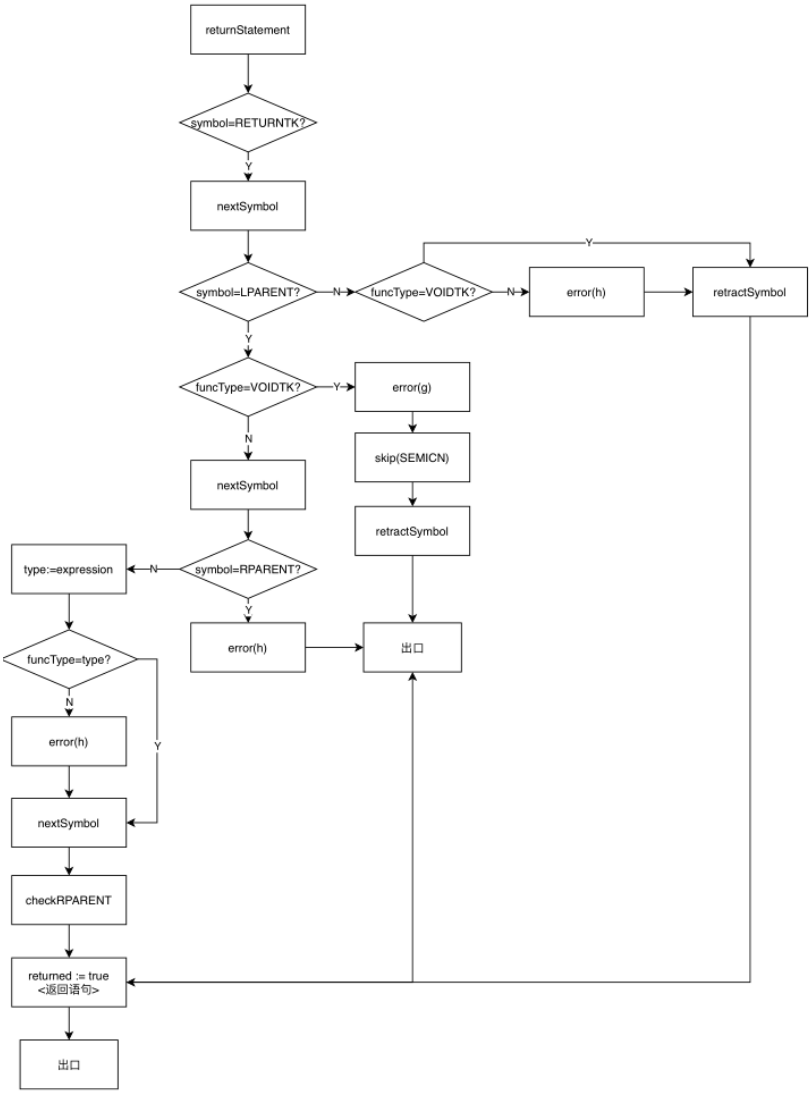

parameterTable
symbol=INTTK/CHARTK?
N
symbol=RPARENT?
N
Y
param push_back(symbol)
retractSymbol
retractSymbol
nextSymbol
<参数表>
symbol=IDENFR?
dup:=checkDupIdenfr
dup=true?
N
symbolTable.insert
nextSymbol
symbol=COMMA?
nextSymbol
retractSymbol
<参数表>
funcDup=true?
Y
N
symbolTable.setFuncParam
出口

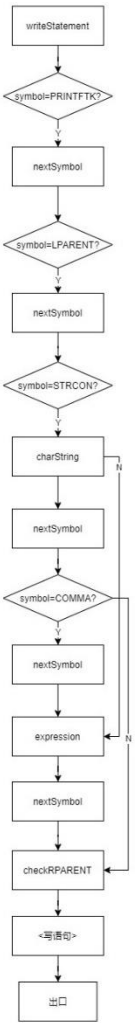

mainFunction
symbol=VOIDTK?
Y
nextSymbol
symbol=MAINTK?
Y
symbolTable.insert
setLevel(1)
nextSymbol
symbol=LPARENT?
Y
nextSymbol
checkRPARENT
nextSymbol
symbol=LBRACE?
Y
nextSymbol
compoundStatement
nextSymbol
checkRBRACE
<主函数>
出口

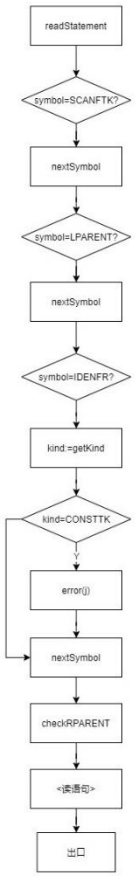

compoundStatement
symbol=CONSTTK?
Y
constDeclare
nextSymbol
N
symbol=
INTTK/CHARTK?
Y
varDeclare
nextSymbol
N
statementList
<复合语句>
出口

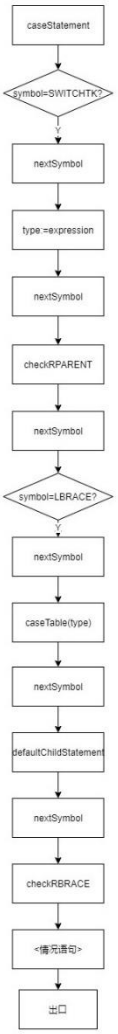

statementList
symbol=RBRACE?
Y
nextSymbol
retractSymbol
<语句列>
出口
N
statement
nextSymbol

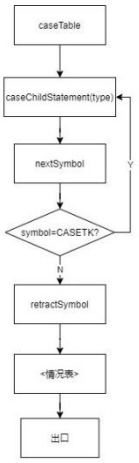

statement
symbol=WHILETK/FORTK?
N
Y
loopStatement
symbol=IFTK?
N
Y
conditional Statement
symbol=SWITCHTK?
N
Y
caseStatement
symbol=IDENFR?
N
Y
nextSymbol
symbol=LPARENT?
N
Y
retractSymbol
callFunction
nextSymbol
checkSEMICN
retractSymbol
assignStatement
<语句>
出口
symbol=SCANFTK?
N
Y
readStatement
symbol=PRINTFTK?
N
Y
writeStatement
nextSymbol
checkSEMICN
symbol=RETURNTK?
N
Y
returnStatement
symbol=LBRACE?
N
Y
nextSymbol
statementList
nextSymbol
checkRBRACE
symbol=SEMICN?
Y

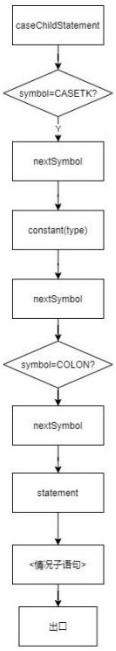

returnStatement
symbol=RETURNTK?
Y
nextSymbol
symbol=LPARENT?
N
funcType=VOIDTK?
N
Y
error(h)
retractSymbol
Y
funcType=VOIDTK?
Y
error(g)
skip(SEMICN)
retractSymbol
N
nextSymbol
symbol=RPARENT?
N
Y
type:=expression
error(h)
出口
funcType=type?
N
Y
error(h)
nextSymbol
checkRPARENT
returned := true
<返回语句>
出口

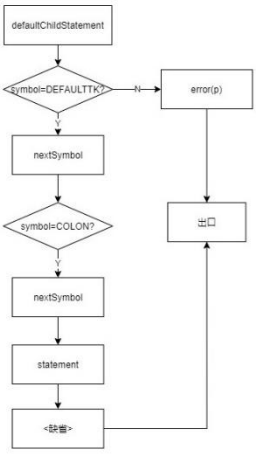

writeStatement
symbol=PRINTFTK?
Y
nextSymbol
symbol=LPARENT?
Y
nextSymbol
symbol=STRCON?
Y
charString
N
nextSymbol
symbol=COMMA?
Y
nextSymbol
expression
N
nextSymbol
checkRPARENT
<写语句>
出口

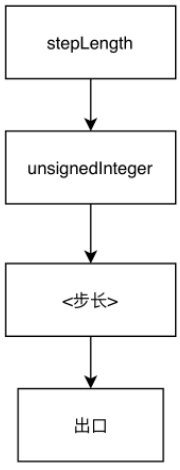

readStatement
symbol=SCANFTK?
nextSymbol
symbol=LPARENT?
nextSymbol
symbol=IDENFR?
kind:=getKind
kind=CONSTTK
Y
error()
nextSymbol
checkRPARENT
<读语句>
出口

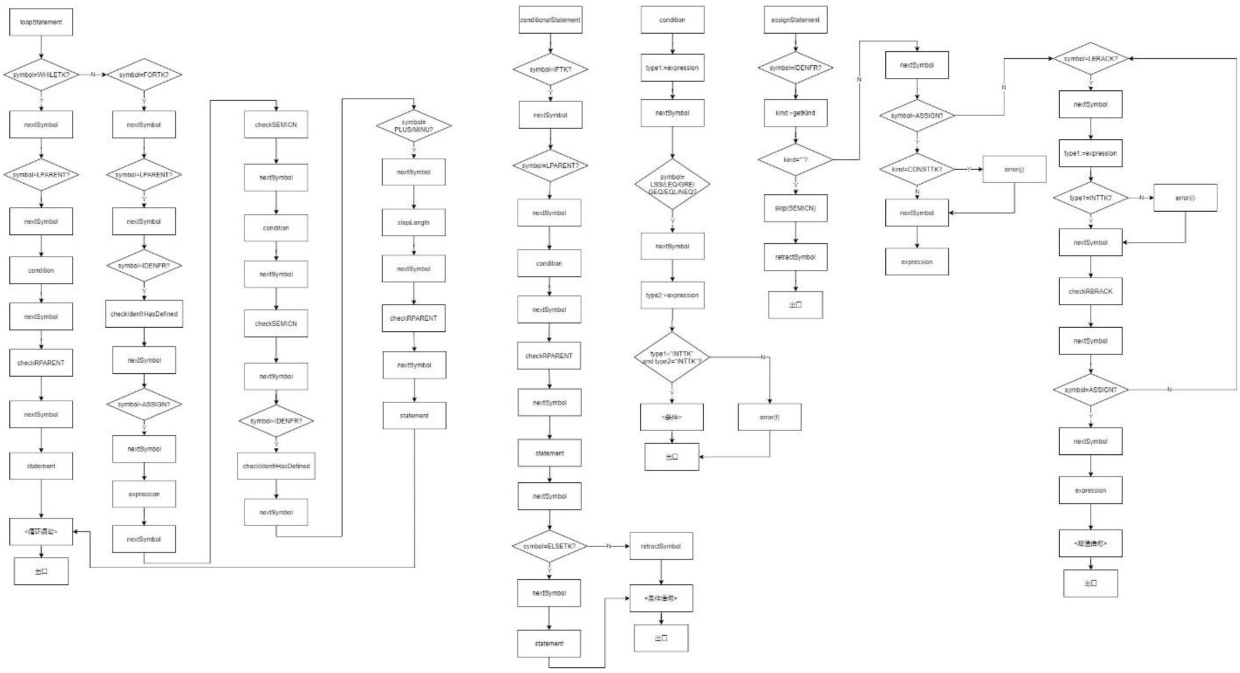

caseStatement
symbol=SWITCHTK?
Y
nextSymbol
type:=expression
nextSymbol
checkRPARENT
nextSymbol
symbol=LBRACE?
Y
nextSymbol
caseTable(type)
nextSymbol
defaultChildStatement
nextSymbol
checkRBRACE
<情况语句>
出口

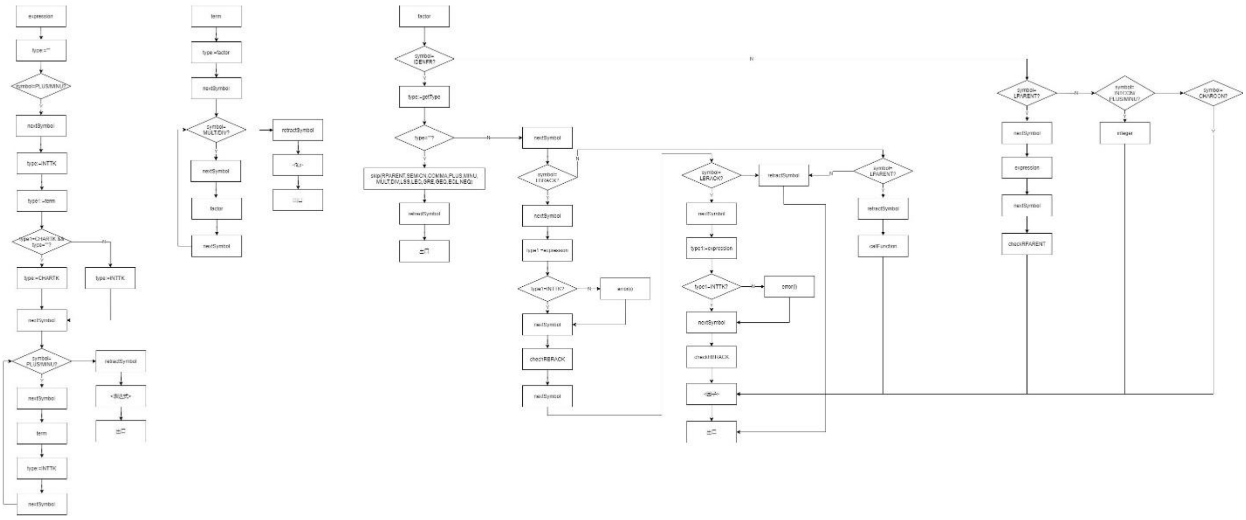

caseTable
caseChildStatement(type)
nextSymbol
symbol=CASETK?
Y
N
retractSymbol
<情况表>
出口

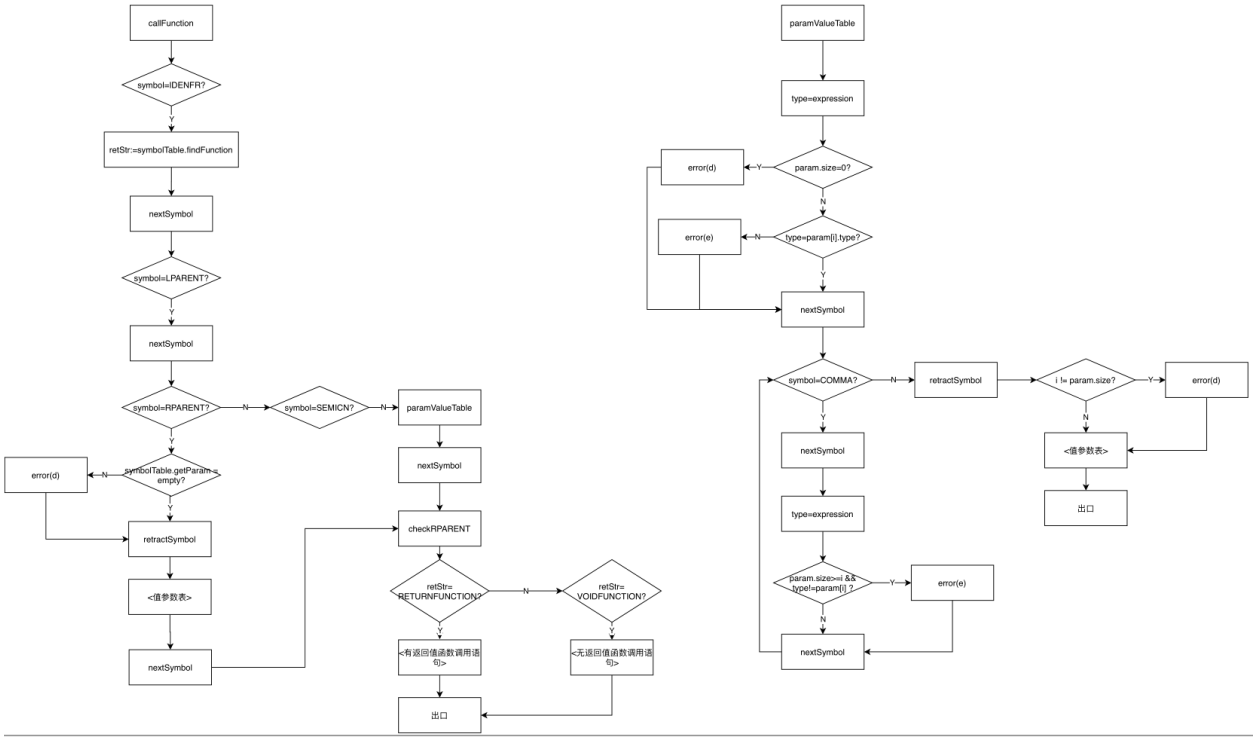

caseChildStatement
symbol=CASETK?
Y
nextSymbol
constant(type)
nextSymbol
symbol=COLON?
Y
nextSymbol
statement
<情况子语句>
出口


defaultChildStatement
symbol=DEFAULTTK?
N
error(p)
Y
nextSymbol
symbol=COLON?
Y
nextSymbol
statement
<缺省>
出口


stepLength
unsignedInteger
<步长>
出口


loopStatement
symbol=WHILETK?
symbol=FORTK?
nextSymbol
symbol=LPARENT?
condition
checkRPARENT
statement
出口
symbol=IDENFR?
checkIdentHasDefined
symbol=ASSIGN?
expression
checkSEMICN
symbol=PLUS/MINU?
stepLength
conditionalStatement
symbol=IFTK?
symbol=ELSETK?
retractSymbol
type1:=expression
type2:=expression
error()
assignStatement
kind:=getKind
skip(SEMICN)
retractSymbol
kind=CONSTTK?
symbol=LBRACK?
type1!=INTTK?
checkRBRACK


expression
term
factor
nextSymbol
retractSymbol
type:=getType
checkRPARENT
callFunction
integer
error()
checkRBRACK


callFunction
symbol=IDENFR?
Y
retStr:=symbolTable.findFunction
nextSymbol
symbol=LPARENT?
Y
nextSymbol
symbol=RPARENT?
N
symbol=SEMICN?
N
paramValueTable
Y
error(d)
N
symbolTable.getParam=empty?
nextSymbol
Y
retractSymbol
checkRPARENT
<值参数表>
retStr=RETURNFUNCTION?
N
retStr=VOIDFUNCTION?
nextSymbol
Y
Y
<有返回值函数调用语句>
<无返回值函数调用语句>
出口
paramValueTable
type=expression
param.size=0?
Y
error(d)
N
type=param[i].type?
N
error(e)
Y
nextSymbol
symbol=COMMA?
N
retractSymbol
i != param.size?
Y
error(d)
Y
nextSymbol
N
<值参数表>
type=expression
出口
param.size>=i &&type!=param[i] ?
Y
error(e)
N
nextSymbol

四、代码生成阶段设计文档

1) 中间代码设计

1. 常量声明

源码：const int i = 10;

中间代码：
（配合符号表信息输出 ）
```
CONST INTTK i = 10
```

2. 变量声明（非数组）

源码： int i = 2;

中间代码：
（配合符号表信息输出 ）
```
VAR INTTK i = 2
```

3. 数组声明

源码：int arr1[3] = {1,2,3};

中间代码：
（配合符号表信息输出 ）
```
ARRAY 1 INIT INTTK arr1 3 1 2 3
```

4. 函数声明

源码：int foo(int a, int b, int c) {}
中间代码：

（配合符号表信息输出 ）
```
FUNCTION foo INTTK
PARA INTTK a
PARA INTTK b
PARA INTTK c
```

5. 函数返回

a. 源码：return;

中间代码：RET

b. 源码：return (expression);

中间代码：RET #expr1

6. 函数调用

源码： foo(i, i+1, 1);
中间代码：

```
PUSH i
PLUS $t2 = i + 1
PUSH $t2
PUSH 1
```

```
CALL foo
```

7. 条件判断

源码: `i + 1 < i * i`

中间代码:

```
PLUS $t2 = i + 1
ASSIGN #expr1 = $t2
MULT $t2 = i * i
ASSIGN #expr2 = $t2
LSS #expr1 #expr2 BF LABEL0
```

8. 设标号（label）

中间代码: `SET LABEL0`

9. 跳转到标号（label）

中间代码:

```
GOTO LABEL0  //直接跳转到LABEL0
LSS #expr1 #expr2 BT LABEL0       //若条件（#expr1 < #expr2）为真，跳转到LABEL0
LSS #expr1 #expr2 BF LABEL0 //若条件（#expr1 < #expr2）为假，跳转到 LABEL0
```

10. 表达式

源码: `i = i + i * i/i;`

中间代码:

```
MULT $t2 = i * i
DIV $t3 = $t2 / i
PLUS $t4 = i + $t3
ASSIGN i = $t4
```

11. 赋值语句

a. 源码:

```
x = 1 + x;
```

中间代码:

```
PLUS $t2 = 1 + x
ASSIGN x = $t2
```

b. 源码:

```
arr1[1] = 3 + x;
```

中间代码:

```
DIM 1 //要访问的数组下标
PLUS $t2 = 3 + x
ASSIGN arr1 = $t2
```

c. 源码:

```
x = arr1[1];
```

中间代码:

```
DIM 1
```

```
LOADARR $t2 arr1 //从数组中取值
ASSIGN x = $t2
```

12. if 语句

源码:

```
if (i < 5) {
	i = i + 1;
}
else if (i > 5) {
	i = i -1;
}
else {
	i = i * 1;
}
```

中间代码:

```
ASSIGN #expr1 = i
ASSIGN #expr2 = 5
LSS #expr1 #expr2 BF LABEL0

PLUS $t2 = i + 1
ASSIGN i = $t2

GOTO LABEL1

SET LABEL0
ASSIGN #expr1 = i
ASSIGN #expr2 = 5
GRE #expr1 #expr2 BF LABEL2

MINU $t2 = i - 1
ASSIGN i = $t2

GOTO LABEL3

SET LABEL2
MULT $t2 = i * 1
ASSIGN i = $t2

SET LABEL3

SET LABEL1
```

13. switch 语句

源码:

```
switch(i) {
	case 1: {
		i = i + 1;
	}
	case 2: {
		i = i - 1;
	}
	default: {
		i = i * 1;
	}
}
```

中间代码:

```
ASSIGN #expr1 = i
EQL #expr1 1 BT LABEL1
EQL #expr1 2 BT LABEL2

GOTO LABEL3

SET LABEL1
PLUS $t2 = i + 1
ASSIGN i = $t2

GOTO LABEL0

SET LABEL2
MINU $t2 = i - 1
ASSIGN i = $t2

GOTO LABEL0

SET LABEL3
MULT $t2 = i * 1
ASSIGN i = $t2

SET LABEL0
```

14. **for 语句**

源码:

```
for(i = 1; i <= 5; i = i + 1) {
	x = x + 1;
}
```

中间代码:

```
ASSIGN i = 1

SET LABEL0
ASSIGN #expr1 = i
ASSIGN #expr2 = 5
LEQ #expr1 #expr2 BF LABEL1

PLUS $t2 = x + 1
ASSIGN x = $t2
PLUS $t2 = i + 1
ASSIGN i = $t2

GOTO LABEL0

SET LABEL1
```

15. **while 语句**

源码:

```
int x = 0;
while(x < 10) {
	x = x + 1;
}
```

中间代码：

```
VAR INTTK x = 0

SET LABEL0
ASSIGN #expr1 = x
ASSIGN #expr2 = 10
LSS #expr1 #expr2 BF LABEL1

PLUS $t2 = x + 1
ASSIGN x = $t2

GOTO LABEL0

SET LABEL1
```

2) 符号表中的辅助信息：

1. `map<string, vector<vector<string>>> funcAttrs;`
   `//(funcName, vector(attrName, attrType, attrSize))`
   存储函数中的变量信息。

2. `map<string, vector<vector<string>>> funcParams;`
   `//(funcName, vector(paramName, paramType))`
   存储函数的参数信息。

3. `map<string, vector<pair<string, int>>> funcCalls;`
   `//(funcname, vector(funcname, paramcount))`
   存储在函数语句执行过程中会调用的函数的信息。

4. `map<string, vector<string>> global_const;`
   `//(name, (type,value))`
   存储全局常量的信息。

5. `map<string, vector<string>> global_var;`
   `//(name, (type, value))`
   存储全局变量（不含数组）的信息。

6. `map<string, vector<string>> global_array;`
   `//(name, (type, length, dim1, dim2))`
   存储全局数组的信息。

7. `map<string, string> global_function;`
   `//(name, return type)`
   存储所有函数的信息。

8. `map<string, string> local_parameter;`
   `//(name, type)`
   存储某函数的参数的信息。

9. `map<string, vector<string>> local_const;`
   `//(name, (type,value))`
   存储某函数的局部常量的信息。

10. `map<string, vector<string>> local_var;`
    `//(name, (type, value))`
    存储某函数的局部变量（不含数组）的信息。

11. map<string, vector<string>> local_array;
    //(name, (type, length, dim1, dim2))
    存储某函数的局部数组的信息。

12. map<string, int> symbolsptable;
    //(name, offset from sp)
    存储某函数的参数、常量、数组、变量在栈上与 stack pointer 的相对位置。

3）程序运行栈的组织方式:

参数 n … 参数 5 $a3 $a2 $a1 $a0	此程序的参数，这一部分属于调用者(caller)的运行栈。 下面的运行栈部分都是被调用者(callee)的运行栈。
$ra	程序开始时需保存这些寄存器，程序返回前恢复。
$fp	程序开始时需保存这些寄存器，程序返回前恢复。
$s7 … $s0	程序开始时需保存这些寄存器，程序返回前恢复。
局部变量 n … 局部变量 0	
额外的辅助空间	运算时可能需要把一些中间值存储起来。
$t7 … $t0	程序调用时需保存这些寄存器，调用结束后恢复。
参数 n … 参数 5 $a3 $a2 $a1 $a0	调用程序时填入的实际参数。

六、代码优化阶段设计文档

只进行了常量传播优化。