



课 程 实 验 报 告

课程名称： 基于高级语言源程序格式处理工具

专业班级 **BD2202** 学 号 **U202217357** 姓 名 吕博文 指导教师 李开 报告日期 **2023** 年 **9** 月 **30** 日

计算机科学与技术学院

目 录

**[1](#_bookmark2)** [问题描述](#_bookmark2)  **[1](#_bookmark3)**

**[2](#_bookmark4)** [程序的总体设计](#_bookmark4)  **[2](#_bookmark5)**

[2.1 词法分析器(lexer)](#_bookmark6) [2](#_bookmark7)

[2.2 语法分析器(parser)](#_bookmark8) [3](#_bookmark1)

[2.3 AST 生成工具](#_bookmark9) [3](#_bookmark1)

**[3](#_bookmark10)** [数据结构和算法详细设计](#_bookmark10)  **[5](#_bookmark11)**

[3.1 词法分析器(lexer)](#_bookmark12) [5](#_bookmark13)

[3.2 语法分析器(parser)](#_bookmark14) [6](#_bookmark15)

[3.3 AST 生成工具](#_bookmark16) [10](#_bookmark17)

**[4](#_bookmark18)** [程序实现](#_bookmark18)  **[12](#_bookmark19)**

**[5](#_bookmark20)** [程序测试及结果分析](#_bookmark20)  **[13](#_bookmark21)**

[5.1 测试文件的内容](#_bookmark22) [13](#_bookmark23)

[5.2 处理后的代码](#_bookmark24) [17](#_bookmark25)

[5.3 报错](#_bookmark26) [22](#_bookmark27)

[5.4 测试程序的 AST](#_bookmark28) [23](#_bookmark29)

**[6](#_bookmark30)** [复杂度的分析](#_bookmark30)  **[30](#_bookmark31)**

**[7](#_bookmark32)** [总结、特色与不足](#_bookmark32)  **[31](#_bookmark33)**

[7.1 高度可视化的抽象语法树](#_bookmark34) [31](#_bookmark35)

[7.2 更完善的错误处理](#_bookmark36) [31](#_bookmark37)

[7.3 可视化系统界面](#_bookmark38) [31](#_bookmark39)

[7.4 完善的图像生成方式](#_bookmark40) [31](#_bookmark41)

[7.5 无法分析语义错误](#_bookmark42) [31](#_bookmark43)

**[8](#_bookmark44)** [参考文献](#_bookmark44)  **[32](#_bookmark45)**

**[9](#_bookmark46)** [附录](#_bookmark46) **[A](#_bookmark46)** [源程序](#_bookmark46)  **[33](#_bookmark47)**

[9.1 总系统](#_bookmark48) [33](#_bookmark49)

[9.2 词法分析器(lexer)](#_bookmark50) [34](#_bookmark51)

[9.3 语法分析器(parser)](#_bookmark52) [59](#_bookmark53)

[9.4 AST 生成工具](#_bookmark54) [143](#_bookmark55)

**[10](#_bookmark56)** [附录](#_bookmark56) **[B](#_bookmark56)** [程序使用说明](#_bookmark56)  **[144](#_bookmark57)**

**1** 问题描述

在计算机科学领域， 抽象语法树(Abstract Syntax Tree，缩写为 AST) 是一 种以树状结构表示源代码语法结构的方法。其中每个树节点对应源代码中的一 个语法成分。与具体语法树(Concrete Syntax Tree) 相比， “抽象” 一词体现在 AST 忽略了源代码细节，专注于核心语法特征。

作为程序的中间表示形式， AST 在程序分析等领域广泛应用， 例如源代码 浏览器、智能编辑器和语言翻译器。

在高级语言源代码处理工具中， AST 是关键的概念。首要步骤是使用巴克 斯范式(BNF) 来形式化定义高级语言的词法规则(即单词的构成规则) 和语法 规则(即构建语句和程序的规则)。接着， 基于形式语言自动机的原理， 进行词 法分析， 以识别出源代码中的所有单词。然后， 通过递归下降语法分析法， 解析 源代码的语法结构， 并生成抽象语法树。最终， 这个抽象语法树可以用于生成格 式化的源代码。

华 中 科 技 大 学 课 程 实 验 报 告

**2** 程序的总体设计

高级语言源程序格式处理工具由 lexer (词法分析器)、parser (语法分析器)、 操作总系统和 AST 图像生成辅助工具构成。它的文件结构如图所示。



图 2- 1 程序的文件结构

**2.1** 词法分析器(**lexer**)

程序的词法分析器(lexer) 采用 C 语言编写， 负责将输入的字符流处理成 词法符号(token) 流并输出至 temp\_token.txt 供语法分析器读取。与此同时， 词 法分析器还可以删除输入文本中的注释，找出输入文本的词法错误。

词法分析器的主程序编译运行时， 会读入 input.txt 中的文本， 将其处理成词

法符号流后写入临时文件 temp\_token.txt, 同时在屏幕上打印出所有词法符号的 类别和值。如果输入文本有词法错误， 词法错误的位置(行号) 和出错原因也会 被打印在屏幕上。

**2.2** 语法分析器(**parser**)

程序的语法分析器(lexer) 采用 C 语言编写， 负责将输入的词法符号(token) 流处理成抽象语法树(AST)， 生成基于 dot 语言的抽象语法树信息并把前序遍 历 AST (这也是文本处理过程) 的结果输出至 result.txt。与此同时， 语法分析器 还可以找出输入文本的语法错误。

语法分析器的主程序编译运行时， 会读入 temp\_token.txt 中的词法符号流， 按照 C 语言子集的文法定义进行递归下降分析， 在分析的过程中进行文本格式 的处理和 dot 代码的生成， 并把上述两个结果输出至 result.txt 和 AST.dot 中， 同 时在屏幕上打印出 AST 的书目表的表示形式。如果输入文本有语法错误， 语法 错误的位置(行号)和出错原因也会被打印在屏幕上。

**2.3** **AST** 生成工具

AST 的生成工具采用 c 语言编写， 用户可以调用 openweb.exe 在线生成图像， 亦可以配置好 graphviz 环境后运行 AST\_creator.bat 文件， 将语法分析器生成的 dot 代码编译成可视化的抽象语法树。下图所示的是简单代码《Hello World!》的 抽象语法树。

输入的代码

1 **#include**<stdio .h>

2 **int** main () 3 {

4 printf ("Hello World !"); 5 }

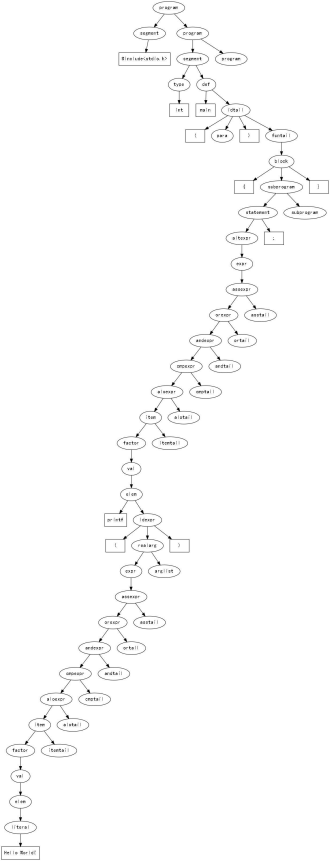


图 2-2 输出的的 AST

**3** 数据结构和算法详细设计

**3.1** 词法分析器(**lexer**)

词法分析器采用“硬编码”的方式实现。下面将介绍词法符号的记录和简要 的词法分析过程。

**3.1.1** 词法符号的记录

本程序定义了一个枚举类型来表示所有 C 语言子集支持的词法符号。 C 语 言子集支持的词法符号主要包括：

1. 标识符

2. 6 种数据类型

3. 不同类型的常量

4. 逻辑运算

5. 关系运算

6. 算术运算

7. 单目运算

8. 逗号、冒号等分隔符

9. 11 种语句的关键字

还包括表示出现了词法错误和文件结束的标签。在进行词法分析的过程中， 采用 TOKEN 结构存储已分析出来的词法符号，其定义如下：

TOKEN 的定义

1 **typedef** **struct** TOKEN**//**定 义**token**。 2 {

3 **int** type;**//**词 法 符 号 的 类 型

4 **char** value [MAXLEN\_OF\_CONTENT];**//**词 法 符 号 的 值

5 **int** loc;**//**词 法 符 号 的 位 置

6 }Token ;

同时我们定义了 7 种词法错误：

1. 字符串缺少双引号

2. 二进制常量无值

3. 十六进制常量无值

4. 字符类型缺少单引号

5. 字符常量无值

6. 逻辑或缺少 |

7. 错误的词法记号

**3.1.2** 词法分析的过程

首先我们进行注释的清除， 当扫描到双斜线时， 表明行注释开始， 开始持续 清除字符直到扫描到换行符。当扫描到连续的斜线和星号时， 表明块注释开始，

开始持续清除字符直到扫描到块注释结束。

当扫描到 #̈ 时 ̈，说明遇到了编译预处理的语句， 这时我们新建一个类型为 “宏”的词法符号并把值设置成这一行的所有字符。

接着我们进行标识符和关键字的处理， 当扫描到下划线或字母时， 说明下一 个词法符号是标识符或关键字。我们继续扫描到第一个不是下划线、数字或字母 的字符， 并存储扫描结果。接着把扫描结果和关键字表比对来确定究竟是关键字 还是标识符。

接着进行常量的判断， 对于数字型常量， 若以 0 开头， 那么它可能是 0 或者 八进制数； 若以 0x 开头， 那么它是十六进制数； 若以0b 开头， 那么它是二进制 数。(标准 C 语言中并没有二进制常量， 这是 C 语言子集的独有的特性)。字符 常量以单引号开头，字符串常量以双引号开头。

最后进行分隔符和运算符的判断，根据扫描到的结果判断即可。

**3.1.3** 词法错误的处理

当词法分析器不能识别输入的词法符号时， 说明输入的文本中出现了词法 错误。本程序使用 lexerr 函数来处理词法错误。 lexerr 函数把输入的词法错误符 号翻译成提示性语句并和错误出现的位置一起打印到屏幕上。

**3.2** 语法分析器(**parser**)

我们采用递归下降的方式， 根据 C 语言子集的文法定义进行语法分析。下 面介绍 C 语言子集的文法定义和递归下降分析的过程。

**3.2.1** **C** 语言子集的文法定义

我们根据如下的文法定义来进行语法分析：

文法定义

1 <program>-><segment><program>

2 <segment>->MAC |EXTERN |<type><def>

3 <type>->INT |VOID |DOUBLE |CHAR |FLOAT

4 <def>->\*ID<init><deflist> |ID<idtail>

5 <init>->=<expr>

6 <deflist>->,<defdata><deflist> |;

7 <idtail>-><varrdef><deflist> |(<para>)<funtail>

8 <defdata>->\*ID<init> |ID<varrdef>

9 <varrdef>->[NUM] |<init>

10 <funtail>->; |<block>

11 <para>-><type><paradata><paralist> |E

12 <funtail>->; |<block>

13 <paradata>->\*ID |ID<paradatatail>

14 <paralist>->,<type><paradata><paralist> |E

15 <paradatatail>->[NUM] |E

16 <block>->{<subprogram>}

17 <expr>-><assexpr>

18 <subprogram>-><type>ID<varrdef><deflist><subprogram> |<statement>< subprogram> |E

19 <assexpr>-><orexpr><asstail>

20 <statement>-><altexpr>; |<whilestat> |<forstat> |<dowhilestat> |< ifstat> |BREAK |CONTINUE |RETURN |GETCHAR

21 <orexpr>-><andexpr><ortail>

22 <asstail>->=<orexpr><asstail> |E

23 <altexpr>-><expr> |E

24 <whilestat>->WHILE (<altexpr>){<block>}

25 <forstat>->FOR(<altexpr>;<altexpr>;<altexpr>){<block>} 26 <dowhilestat>->DO<block>WHILE (<altexpr>)

27 <ifstat>->IF(<expr>)<block><elsestat>

28 <andexpr>-><cmpexpr><andtail>

29 <ortail>->OR<andexpr><ortail> |E

30 <elsestat>->ELSE<block> |E

31 <cmpexpr>-><aloexpr><cmptail>

32 <andtail>->AND<cmpexpr><andtail> |E

33 <aloexpr>-><item><alotail>

34 <cmptail>->< |> |>= |<= | != |==<aloexpr><cmptail> |E

35 <item>-><factor><itemtail>

36 <alotail>->- |+<item><alotail> |E

37 <factor>->- |& |-- |++<factor> |<val>

38 <itemtail>->\* |% |/<factor><itemtail> |E

39 <val>-><elem>-- |++

40 <elem>->ID<idexpr> |(<expr>) |<literal>

41 <idexpr>->[<expr>] |(<realarg>) |E

42 <literal>->NUM |CHAR |STR

43 <realarg>-><expr><arglist> |E

44 <arglist>->,<expr><arglist> |E

**3.2.2** 递归下降分析的过程

递归下降是对一棵树进行前序遍历的一种方式。我们以文法定义的第一条 规则为例来介绍递归下降。

文法定义的第一条规则

1 <program>-><segment><program>

它的含义是从抽象语法树的 program 节点可以进入 segment 节点， 也可以回到 program 节点(右递归)。我们的程序是这样写的

递归下降的例子

1 **void** program () 2 {

3 segment ();

4 program (); 5 }

这样在运行时会先进入表示第一个子节点的函数， 在进入表示第二个子节点的 函数， 以此类推， 就可以实现抽象语法树的前序遍历。当函数运行至表示终结符 的节点时， 我们会判断当前的词法符号和需要的词法符号是否匹配， 如果不匹配

则需要报告相应的语法错误。下面的代码是一个处理终结符的例子， 表示函数或 变量类型的分析(有一些删减)：

type 函数中的部分代码

1 **void** type () 2 {

3 **switch** (nexttype)**//** 匹 配 词 法 符 号 4 {

5 **case** INT:**//INT** 匹 配 成 功

6 gonext ();

7 **break** ;

8 **case** CHAR:**//CHAR** 匹 配 成 功

9 gonext ();

10 **default** :

11 synerr (nexttype ,3,INT ,CHAR ,VOID);**//**词 法 符 号 不 匹 配 ， 进 入 语 法 错 误 处理 函 数

12 **break** ; 13 }

14 }

**3.2.3** **dot** 代码的生成和书目表形式的 **AST** 的打印

在前序遍历获得抽象语法树的过程中， 我们可以插入控制 dot 代码生成和打 印书目表形式的抽象语法树的代码。

在最后画出的抽象语法树中， 我们以长方形节点表示终结符， 以椭圆形节点 表示非终结符。同时， 在节点上标出这个节点的名称。根据这些要求， 我们在程 序运行到一个节点的函数内时， 首先判断这个节点是否是终结符节点， 如果是， 那么把这个节点设置成长方形的， 并把这个节点的名称设置成对应的终结符的 类型名； 如果不是， 那么把这个节点设置成椭圆形的， 并把这个节点的名称设置

成对应的非终结符的名称，然后在这个节点和所有的子节点连一条有向边。

在书目表形式的打印中， 我们使用一个全局变量 layer 来维护遍历到的 AST 的层数。使用 in 函数和 out 函数使 layer 变化， 表示从当前层向下进入下一层或 向上回溯上一层。我们使用 print\_blank 函数来打印书目表中的空格， print\_blank 读取 layer 的值，并据此打印出不同数量的制表位。

**3.2.4** 语法错误的处理

当语法分析器获得了不能匹配的词法符号时， 说明发生了语法错误。本程 序使用 synerr 和 printerr 两个函数来处理语法错误。 printerr 的功能相对简单， 即 输入的语法错误符号翻译成提示性语句并和错误出现的位置一起打印到屏幕上。 synerr 函数是一个可变参数的函数， 它读取当前出错的词法符号和所有期望中的 词法符号。当出错的词法符号的所有可能的前一个符号包含期望中的词法符号 之一时，认为对应的期望中的词法符号缺失，否则认为词法符号错误匹配。

**3.3** **AST** 生成工具

语法分析器运行后会生成名为 AST.dot 的 dot 文件。我们可以调用 open- web.exe 在线编译这个文件， 亦可以配置好 graphviz 环境后运行 AST\_creator.bat 文件并生成最终的 AST.png 。其代码如下：

openweb.exe

1 **#include** "stdio .h"

2 **#include** "stdlib .h"

3 **#include**<windows .h>

4 **#include**<string .h>

5 **void** open\_web (**char** \*website) 6 {

7 ShellExecuteA (0,"open" , website ,0 ,0 ,1);

8 }

9

10 **int** main(**int** argc ,**char** \*argv []) 11 {

12 HWND hwnd;

13 hwnd=FindWindow ("ConsoleWindowClass" ,NULL);

14 ShowWindow (hwnd ,SW\_HIDE);

15 open\_web ("http ://dreampuf .github .io/GraphvizOnline");

16 **return** 0;

17 }

AST\_creator.bat

1 dot -Tpng AST .dot -o AST .png

**4** 程序实现

本程序在 windows11 操作系统下的 devC++ 编辑器中采用C 语言开发完成。

测试的硬件环境为处理器 AMD Ryzen 7 6800H ，内存 16G。

在测试本程序时， 我们首先运行主系统.exe ，选择 1 进行语义分析， 再选 择 2 进行词法分析。最后在 windows11 操作系统中运行 AST\_creator.bat 生产 AST.png。用户亦可以运行 openweb.exe，复制 AST.dot 文件进行在线编译， 下载 并得到 AST.png。

需要注意的是， 使用 openweb.exe 需要配置好网络环境才能够获得 AST 图， 但是系统本地的编译环境并不影响处理后的文本的输出。

**5** 程序测试及结果分析

我们用 5 个测试文件进行了程序的测试。测试文件覆盖了绝大部分 C 语言 子集支持的词法符号和语法格式， 同时， 这些文件的缩进格式都有待改善。这 5 个测试程序分别是：

1. Hello World!

2. 表达式和复杂注释

3. 各种定义和函数的调用

4. 各种语句的嵌套

5. 样例

**5.1** 测试文件的内容

Hello World!

1 **#include**<stdio .h>

2 **int** main () 3 {

4 printf ("Hello World !"); 5 }

表达式和复杂注释

1 **//**表 达 式和 复 杂注 释

2 **#include**<stdio .h>

3 **int** main () 4 {

5 **int** i,j;

6 i=12;

7 i=021;

8 i=0b1;

9 i=(1%1)+2-3\*4/5;

10 j=**/\*** **(1%1)+2-3\*\*/**4/5; 11 **///\*\*\*/**

12 **/\*//\*/**

13 **return** 0;

14 }

各种定义和函数的调用

1 **/\*** 各 种定 义 和 函 数 的 调 用 **\*/**

2 **#include**<stdio .h>

3 **#include**<stdlib .h>

4 **#define** A 10

5 **#define** F(a,b) a+b

6 **int** func1 (**int** a,**int** b);

7 **int** func2 (**int** a,**int** b) 8 {**return** b+a;}

9 **int** main () 10 {

11 **int** i=0,j=0,k;

12 **float** m=0 .5;

13 **double** n;

14 **int**\*p=&i;

15 **int**\*q;

16 **int** arr [15];

17 k=func1 (i,j)+func2 (i,j);

18 **return** 0;

19 }

20 **int** func1 (**int** a,**int** b) 21 {

22 **int** c=a+b;

23 **return** c; 24 }

各种表达式的嵌套

1 **/\*** 各 种表 达 式 的 嵌 套 **\*/**

2 **#include**<stdio .h>

3 **int** main ()

4 {**int** i=0,j=0,k=0;

5 **//if**

6 **if** (i==0)

7 {

8 **if** (j==0) 9 {

10 k=0; 11 }

12 k=0;

13 }

14 **if** (j==0){k=0;}

15 **else** **if** (j!=0){k=0;}

16 **else**{j=0;} 17 **//while**

18 **while** (i!=0) 19 {

20 **while** (j!=0) 21 {

22 k=0; 23 }

24 k=0;

25 }

26 **while** (j!=0){k=0;} 27 **//for**

28 **for**(i=0;i<2;i++) 29 {

30 **for**(**int** m=4;m>2;m--) 31 {

32 i=0;

33 **continue** ;

34 }

35 }

36 **for**(i=0;i<=2;i++){i=0;} 37 **//do-while**

38 **do** 39 {

40 **do** 41 {

42 k=0; 43 }**while** (j!=0);

44 k=0; 45 }**while** (i!=0);

46 **do**{k=0;}

47 **while** (j!=0); 48 **//switch-case**

49 **switch** (i) 50 {

51 **case** 1:

52 i=0;

53 **break** ;

54 **case** 2:

55 i=1;

56 **break** ;

57 **case** 3:

58 **switch** (i){**case** 1:**break** ;}

59 **default** :

60 **break** ;

61 }

62 **switch** (i){**case** 1:**break** ;}

63 **return** 0; 64 }

样例

1 **int** i,j;

2 **int** fun(**int** a, **float** b) 3 {

4 **int** m;

5 **if** (a>b)

6 m=a;

7 **else**

8 m=b;

9 **return** m;

10 }

11 **float** x,y;

**5.2** 缩进处理后的代码

Hello World!

1 **#include**<stdio .h>

2 **int** main ( )

3

4 {

5 printf ( "Hello World !" ) ; 6 }

表达式和复杂注释

1 **#include**<stdio .h>

2 **int** main ( )

3

4 {

5 **int** i , j ;

6 i = 12 ;

7 i = 21 ;

8 i = 1 ;

9 i = ( 1 % 1 ) + 2 - 3 \* 4 / 5 ;

10 j = 4 / 5 ;

11 **return** 0 ;

12 }

各种定义和函数的调用

1 **#include**<stdio .h>

2 #include<stdlib .h>

3 #define A 10

4 #define F(a,b) a+b

5 **int** func1 ( **int** a , **int** b ) ;

6 **int** func2 ( **int** a , **int** b )

7

8 {

9 **return** b + a ;

10 }

11 **int** main ( ) 12

13 {

14 **int** i = 0 , j = 0 , k ;

15 **float** m = 0 .5 ;

16 **double** n ;

17 **int** \* p = & i ;

18 **int** \* q ;

19 **int** arr [ 15 ] ;

20 k = func1 ( i , j ) + func2 ( i , j ) ;

21 **return** 0 ;

22 }

23 **int** func1 ( **int** a , **int** b ) 24

25 {

26 **int** c = a + b ;

27 **return** c ;

28 }

各种表达式的嵌套

1 **#include**<stdio .h>

2 **int** main ( )

3

4 {

5 **int** i = 0 , j = 0 , k = 0 ;

6 **if** ( i == 0 )

7

8 {

9 **if** ( j == 0 )

10

11 {

12 k = 0 ;

13 }

14 k = 0 ; 15 }

16 **if** ( j == 0 )

17

18 {

19 k = 0 ; 20 }

21 **else**

22 **if** ( j != 0 ) 23

24 {

25 k = 0 ; 26 }

27 **else**

28

29 {

30 j = 0 ; 31 }

32 **while** ( i != 0 ) 33

34 {

35 **while** ( j != 0 ) 36

37 {

38 k = 0 ; 39 }

40 k = 0 ; 41 }

42 **while** ( j != 0 ) 43

44 {

45 k = 0 ; 46 }

47 **for** ( i = 0 ; i < 2 ; i ++ )

48

49 {

50 **for** ( **int** m = 4 ; m > 2 ; m -- ) 51

52 {

53 i = 0 ;

54 **continue** ; 55 }

56 }

57 **for** ( i = 0 ; i < = = 2 ; i ++ ) 58

59 {

60 i = 0 ; 61 }

62 **do** 63 {

64 **do** 65 {

66 k = 0 ; 67 }

68 **while** ( j != 0 ) ;

69 k = 0 ;

70 }

71 **while** ( i != 0 ) ;

72 **do** 73 {

74 k = 0 ; 75 }

76 **while** ( j != 0 ) ;

77 **switch** ( i )

78

79 {

80 **case** 1 :

81 i = 0 ;

82 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

83 **case** 2 :

84 i = 1 ;

85 **break** ;

86 **case** 3 :

87 **switch** ( i ) 88

89 {

90 **case** 1 :

91 **break** ; 92 }

93 **default** :

94 **break** ; 95 }

96 **switch** ( i )

97

98 {

99 **case** 1 :

100 **break** ; 101 }

102 **return** 0 ;

103 }

样例

1 **int** i , j ;

2 **int** fun ( **int** a , **float** b ) 3

4 {

5 **int** m ;

6 **if** ( a > b )

7 m = a ;

8 **else**

9 m = b ;

10 **return** m ; 11 }

12 **float** x , y ;

华 中 科 技 大 学 课 程 实 验 报 告

**5.3** 报错

为了使错误明显，我们选用较短的程序进行报错测试。

词法错误：字符串缺少右双引号

1 **#include**<stdio .h>

2 **int** main () 3 {

4 printf ("Hello World !) ; 5 }



图 5- 1 字符串缺少右双引号的报错

词法错误：错误的词法符号

1 **#include**<stdio .h>

2 **int** main ()

3 {

4 $

5 printf ("Hello World !");

6 }



图 5-2 错误的词法符号的报错

语法错误：类型缺失

1 **#include**<stdio .h>

2 main () 3 {

4 printf ("Hello World !"); 5 }

语法错误：右小括号缺失

1 **#include**<stdio .h>

2 **int** main(



华 中 科 技 大 学 课 程 实 验 报 告

图 5-3 类型缺失的报错

3 {

4 printf ("Hello World !");

5 }



图 5-4 右小括号缺失的报错

语法错误：分号缺失

1 **#include**<stdio .h>

2 **int** main () 3 {

4 printf ("Hello World !") 5 }

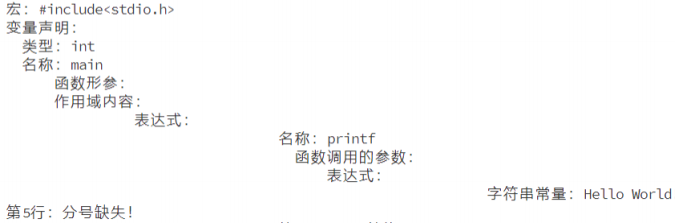


图 5-5 分号缺失的报错

**5.4** 测试程序的 **AST**

各种表达式的嵌套的图过于庞大，是分片插入的，完整的图请见附件。

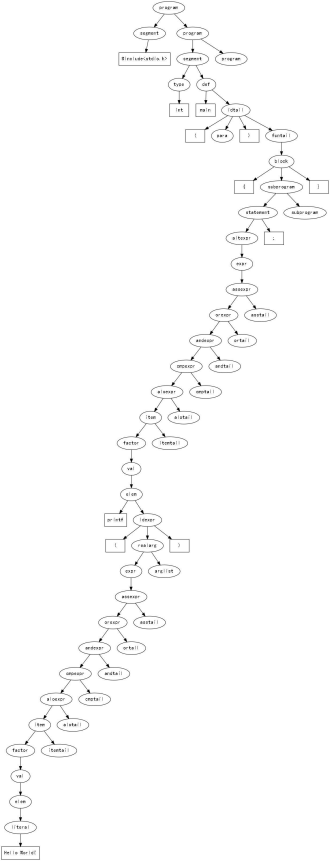


图 5-6 Hello World!

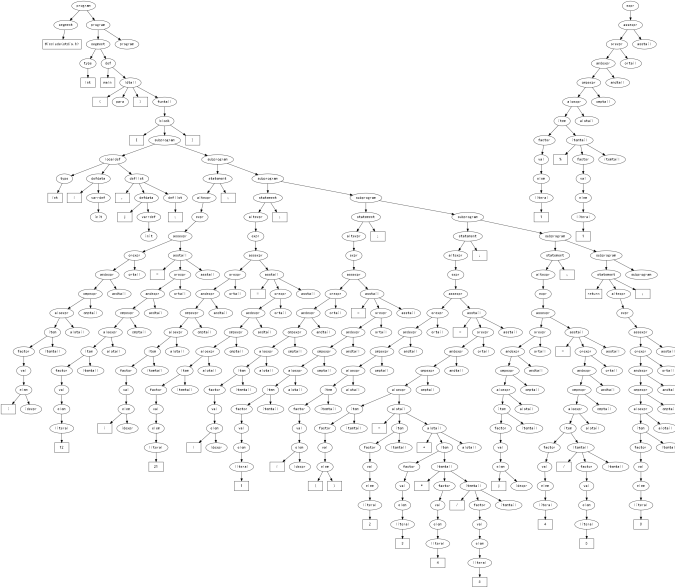


图 5-7 表达式和复杂注释

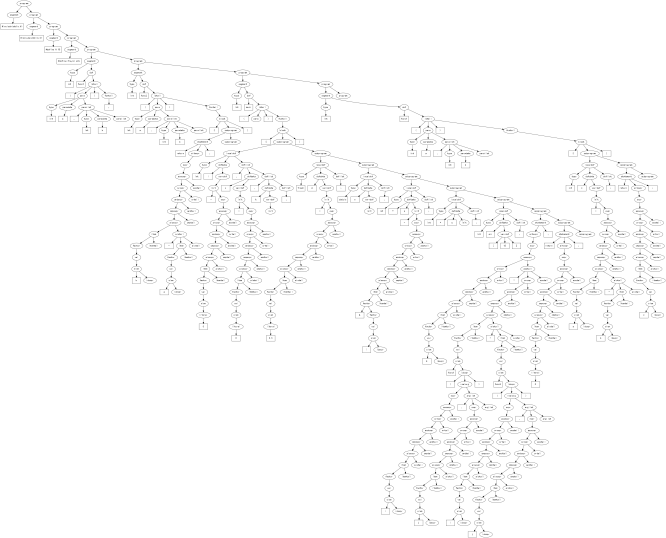
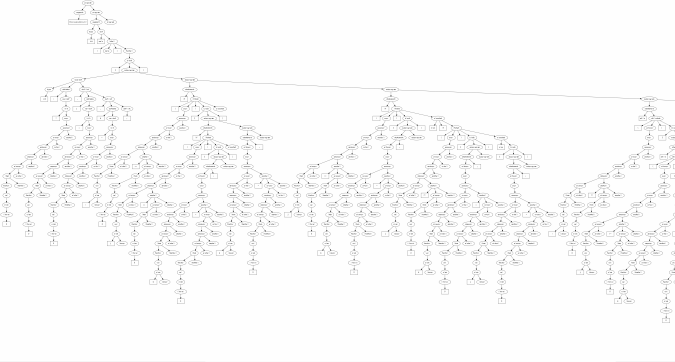
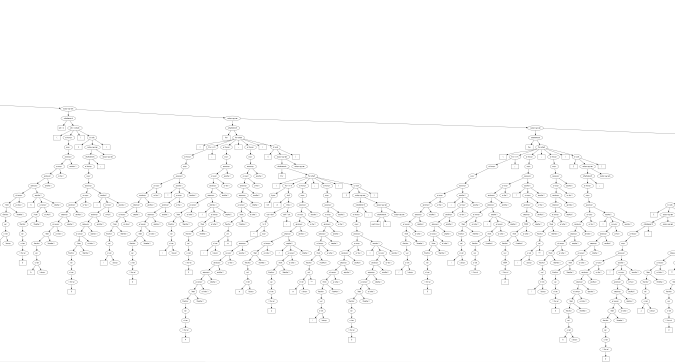


图 5-8 各种定义和函数的调用

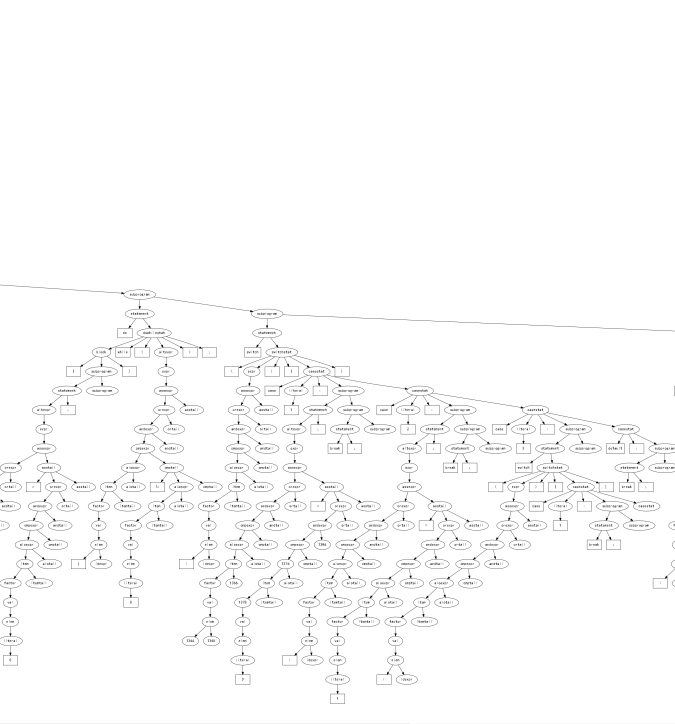




华 中 科 技 大 学 课 程 实 验 报 告

图 5-9 各种表达式的嵌套

图 5- 10 各种表达式的嵌套



华 中 科 技 大 学 课 程 实 验 报 告

图 5- 11 各种表达式的嵌套

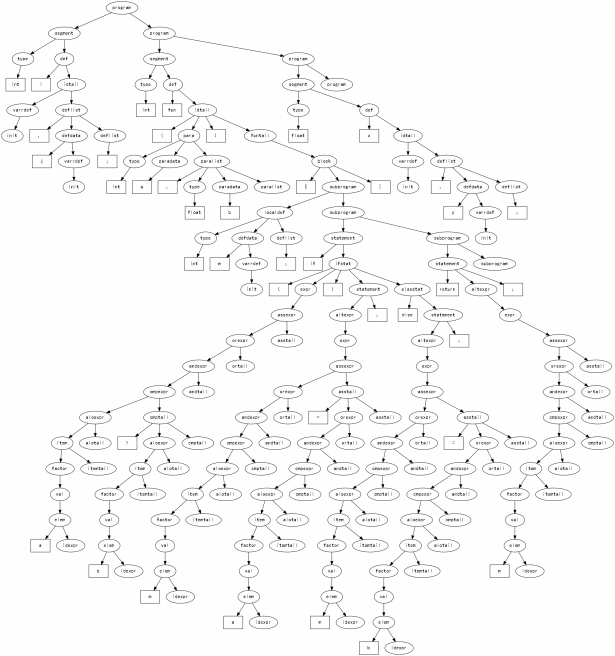


图 5- 12 样例

**6** 复杂度的分析

本程序将特定的值或参数直接写入代码中，从而进行词法分析，并采用递

归下降的方式进行语法分析，时间复杂度是线性的。

**7** 总结、特色与不足

**7.1** 高度可视化的抽象语法树

树的表示方法有书目表法、前中序遍历法等。但这些方法普遍存在着可视化 程度低、不直观等缺点。对于很复杂的抽象语法树， 这种缺点使用户难以理解其 结构。本程序根据语法树的结构， 使用 Graphviz 生成 AST.png 图像， 实现了对 抽象语法树的直观展现。

**7.2** 更完善的错误处理

实验要求中对于错误处理仅要求输出词法错误、语法错误的行号， 而不要求 输出错误原因， 这对我们修改错误增加了很多不便。本程序除了可以输出词法 错误、语法错误的行号之外， 还可以输出错误原因， 这样可以使修改错误更为准 确、容易。

**7.3** 可视化系统界面

本程序包含可视化的操作界面， 列举了词法分析、语法分析、缩进编排等操 作，便于用户使用。同时可以输出内容到屏幕，避免反复切换程序查看结果。

**7.4** 完善的图像生成方式

本程序考虑了用户有 Graphviz 本地环境以未配置环境的两种情况： 用户既 可以在本地配置好后运行 creator\_AST.bat 在源目录生产 AST 图像， 也可以运行 openweb.exe 进行在线下载图像。

**7.5** 无法分析语义错误

对于部分词法、语法均正确但语义出错的代码， 本程序无法进行合理的分 析。

**8** 参考文献

[1] 王生原， 董渊， 张素琴， 吕映芝等. 编译原理(第 3 版) . 北京： 清华大

学出版社. 前 4 章

[2] 严蔚敏等. 数据结构 (C 语言版). 北京：清华大学出版社

[3] Graphviz 使用说明 <https://graphviz.org/documentation/>

**9** 附录 **A** 源程序

**9.1** 总系统

总系统

1 #include"lexer\lexer\_main .c"

2 #include"parser\parser\_main .c"

3 **int** main () {

4 **int** op = 1;**//**初 始 化 选 项

5 FILE \*file;**//**定 义 文 件 指 针

6 **while** (op) {

7 system ("cls");

8 printf ("------------------欢 迎 使 用 基 于 高

级 语 言 源 程 序 格 式处 理 工 具

------------------\n");

9 printf ("1. 词 法 分 析\n2 . 语 法 分 析\n3 . 缩 进 编 排\n0 . 退 出 程 序\n请 选 择 ： ");

10 choose : scanf ("%d" , &op); 11 **//system** **("cls")** **;**

12 **if** (op==1){

13 lexer\_main ();

14 printf ("\n按 任 意 键 继 续 . . .\n");

15 getchar ();

16 **goto** choose ;

17 }**//**进 行 词 法 分 析

18 **if** (op==2){

19 parser\_main ();

20 printf ("\n按 任 意 键 继 续 . . .\n");

21 getchar ();

22 **goto** choose ; 23 }**//**进 行 语 法 分 析

24 **if** (op==3){

25 file = fopen ("result .txt" , "r");

**//** **"r"** 表 示 以 只 读 方 式打 开 文 件

26 **if** (file == NULL) {

27 printf ("无 法 打 开 文 件\n"); 28 }

29 **char** line [100]; **//** 假 设 每 行 不 超 过

**100**个 字 符

30 **while** (fgets (line , **sizeof** (line), file)) {

31 printf ("%s" , line); 32 }**//**打 印 缩 进 编 排 后 的 文 件

33 fclose (file);

34 printf ("\n按 任 意 键 继 续 . . .\n");

35 getchar ();

36 **goto** choose ;

37 }**//**进 行 缩 进 编 排

38 **if** (op==0){

39 **return** 0; 40 }

41 } **//** **end** **of** **while**

42 **//return** **0;**

43 }

**9.2** 词法分析器(**lexer**)

lexer\_datastructure

1 **#ifndef** \_\_LEXER\_DATASTRUCTURE\_\_

2 **#define** \_\_LEXER\_DATASTRUCTURE\_\_

3 **typedef** **struct** TOKEN**//**定 义**token** **.**

4 {

5 **int** type;

6 **char** value [MAXLEN\_OF\_CONTENT];

7 **int** loc; 8 }Token ;

9

10 **enum** Tag**//**词 法 符 号 的 标 签

11 {

12 ERR ,**//**错 误。

13 END ,**//**文 件 结 束。

14 ID ,**//**标 识 符。

15 INT ,CHAR ,FLOAT ,DOUBLE ,VOID ,EXTERN ,**//**数 据 类 型。

16 NUM ,**//**数 字 常 量。

17 CH ,**//** 字 符 常 量。

18 STR ,**//** 字 符 串。

19 NOT ,LEA ,**//**！**&**

20 ADD ,SUB ,MUL ,DIV ,MOD ,**//+-\*/%**

21 INC ,DEC ,**//++--**

22 GT ,GE ,LT ,LE ,EQU ,NEQU ,**//>>=<<==!=**

23 AND ,OR ,**//&&** **|** **|**

24 LPAREN ,RPAREN ,**//** ()

25 LBRACK ,RBRACK ,**//**【 】

26 LBRACE ,RBRACE ,**//** {}

27 COMMA ,COLON ,SEMICON ,**//** ， ： ；

28 ASSIGN ,**//=**

29 IF ,ELSE ,SWITCH ,CASE ,DEFAULT ,WHILE ,DO ,FOR ,BREAK ,CONTINUE , RETURN ,**//** 关 键 字。

30 MAC**//** 宏。

31 };

32

33 **enum** LEXERROR**//**词 法 错 误 的 标 签 34 {

35 STR\_NO\_R\_QUTION ,**//** 字 符 串 缺 少 右 双 引 号。

36 NUM\_BIN\_TYPE ,**//** 二 进 制 常 量 无 值。

37 NUM\_HEX\_TYPE ,**//** 十 六 进 制 常 量 无 值。

38 CHAR\_NO\_R\_QUTION ,**//** 字 符 类 型 缺 少 右 单 引 号。

39 CHAR\_NO\_DATA ,**//** 字 符 常 量 无 值。

40 OR\_NO\_PAIR ,**//**逻 辑 或 缺 少 **|** 。

41 TOKEN\_NO\_EXIST**//**错 误 的 词 法 记 号。

42 };

43 **#endif**

lexer\_def

1 **#ifndef** \_\_LEXER\_DEF\_\_

2 **#define** \_\_LEXER\_DEF\_\_

3 **#include**<stdio .h>

4 **#include**<stdlib .h>

5 **#include**<ctype .h>

6 **#include**<string .h>

7 **#define** MAXLEN\_OF\_CONTENT 50

8 **#define** MAXLEN 1000

9 **#include**"lexer\_datastructure .h"

10 **#include**"lexer\_head .h"

11 **#include**"lexer\_func .c"

12 **#endif**

lexer\_func

1 **#ifndef** \_\_LEXER\_FUNC\_\_

2 **#define** \_\_LEXER\_FUNC\_\_

3 **void** wordscanner (FILE\*fp ,**char** file []) 4 {

5 **int** i=0;

6 **char** c;

7 **while** ((c=fgetc (fp))!=EOF) 8 {

9 file[i]=c;

10 i++; 11 }

12 file[i]= '\0 ' ;

13 }

14 **int** nextstep (**char** file [],**int** cnt ,**char**\*present ,**char**\*next) 15 {

16 \*present=\*next;

17 **if** (file[cnt]!= '\0 ') 18 {

19 \*next=file [1+cnt];

20 **return** 1; 21 }

22 **else**{**return** 0;}

23 }

24 **void** token\_print\_to\_file (FILE\*fp ,Token tokenarray []) 25 {

26 **int** i=0,type=0;

27 **if** (error==1){**return** ;}**//**有 词 法 错 误 ， 直 接 返 回。

28 **while** ((type=tokenarray [i++] .type)!=END) 29 {

30 fprintf (fp ,"%d\n" ,tokenarray [i-1] .type);

31 **if** (type==NUM | |type==STR | |type==ID | |type==CH | |type ==MAC){fprintf (fp ,"%s\n" ,tokenarray [i-1] .value );}

32 fprintf (fp ,"%d\n" ,tokenarray [i-1] .loc); 33 }

34 fprintf (fp ,"%d\n" ,tokenarray [i-1] .type);

35 fprintf (fp ,"%d" ,tokenarray [i-1] .loc);

36 }

37 **void** token\_print\_to\_stdout (Token tokenarray []) 38 {

39 **int** i=0,type=0;

40 **if** (error==1){**return** ;}**//**有 词 法 错 误 ， 直 接 返 回。

41 **while** ((type=tokenarray [i] .type)!=END) 42 {

43 i++;

44 printf (" 第%d个 词 法 符 号 ： \n" ,i);

45 **switch** (type) 46 {

47 **case** ID:

48 printf ("\t种 别 ： 标 识 符\n");

49 printf ("\t 内 容 ： %s\n" ,tokenarray [ i-1] .value);

50 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

51 **break** ;

52 **case** INT:

53 printf ("\t种 别 ： 整 型 类 型\n");

54 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

55 **break** ;

56 **case** CHAR:

57 printf ("\t种 别 ： 字 符 型 类 型\n");

58 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

59 **break** ;

60 **case** FLOAT :

61 printf ("\t种 别 ： 单 精 度 浮 点 型 类 型\ n");

62 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

63 **break** ;

64 **case** DOUBLE :

65 printf ("\t种 别 ： 双 精 度 浮 点 型 类 型\ n");

66 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

67 **break** ;

68 **case** VOID:

69 printf ("\t种 别 ： 空 类 型\n");

70 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

71 **break** ;

72 **case** EXTERN :

73 printf ("\t种 别 ： 全 局 变 量 声 明\n");

74 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

75 **break** ;

76 **case** NUM:

77 printf ("\t种 别 ： 数 值 常 量\n");

78 printf ("\t 内 容 ： %s\n" ,tokenarray [ i-1] .value);

华 中 科 技 大 学 课 程 实 验 报 告

79 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

80 **break** ;

81 **case** CH:

82 printf ("\t种 别 ： 字 符 型 常 量\n");

83 printf ("\t 内 容 ： %s\n" ,tokenarray [ i-1] .value);

84 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

85 **break** ;

86 **case** STR:

87 printf ("\t种 别 ： 字 符 串 常 量\n");

88 printf ("\t 内 容 ： %s\n" ,tokenarray [ i-1] .value);

89 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

90 **break** ;

91 **case** NOT:

92 printf ("\t种 别 ： 逻 辑 非\n");

93 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

94 **break** ;

95 **case** LEA:

96 printf ("\t种 别 ： 取 地 址\n");

97 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

98 **break** ;

99 **case** ADD:

100 printf ("\t种 别 ： 加 法 运 算\n");

101 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

102 **break** ;

103 **case** SUB:

104 printf ("\t种 别 ： 减 法 运 算\n");

105 printf ("\t位 置 : 第%d行\n" ,

华 中 科 技 大 学 课 程 实 验 报 告

tokenarray [i-1] .loc);

106 **break** ;

107 **case** MUL:

108 printf ("\t种 别 ： 乘 法 运 算\n");

109 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

110 **break** ;

111 **case** DIV:

112 printf ("\t种 别 ： 除 法 运 算\n");

113 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

114 **break** ;

115 **case** MOD:

116 printf ("\t种 别 ： 取 余 运 算\n");

117 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

118 **break** ;

119 **case** INC:

120 printf ("\t种 别 ： 自 增 运 算\n");

121 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

122 **break** ;

123 **case** DEC:

124 printf ("\t种 别 ： 自 减 运 算\n");

125 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

126 **break** ;

127 **case** GT:

128 printf ("\t种 别 ： 大 于\n");

129 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

130 **break** ;

131 **case** GE:

132 printf ("\t种 别 ： 大 于 等 于\n");

133 printf ("\t位 置 : 第%d行\n" ,

华 中 科 技 大 学 课 程 实 验 报 告

tokenarray [i-1] .loc);

134 **break** ;

135 **case** LT:

136 printf ("\t种 别 ： 小 于\n");

137 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

138 **break** ;

139 **case** LE:

140 printf ("\t种 别 ： 小 于 等 于\n");

141 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

142 **break** ;

143 **case** EQU:

144 printf ("\t种 别 ： 等 于\n");

145 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

146 **break** ;

147 **case** NEQU:

148 printf ("\t种 别 ： 不 等 于\n");

149 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

150 **break** ;

151 **case** AND:

152 printf ("\t种 别 ： 逻 辑 与\n");

153 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

154 **break** ;

155 **case** OR:

156 printf ("\t种 别 ： 逻 辑 或\n");

157 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

158 **break** ;

159 **case** LPAREN :

160 printf ("\t种 别 ： 左 小 括 号\n");

161 printf ("\t位 置 : 第%d行\n" ,

华 中 科 技 大 学 课 程 实 验 报 告

tokenarray [i-1] .loc);

162 **break** ;

163 **case** RPAREN :

164 printf ("\t种 别 ： 右 小 括 号\n");

165 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

166 **break** ;

167 **case** LBRACK :

168 printf ("\t种 别 ： 左 中 括 号\n");

169 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

170 **break** ;

171 **case** RBRACK :

172 printf ("\t种 别 ： 右 中 括 号\n");

173 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

174 **break** ;

175 **case** LBRACE :

176 printf ("\t种 别 ： 左 大 括 号\n");

177 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

178 **break** ;

179 **case** RBRACE :

180 printf ("\t种 别 ： 右 大 括 号\n");

181 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

182 **break** ;

183 **case** COMMA :

184 printf ("\t种 别 ： 逗 号\n");

185 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

186 **break** ;

187 **case** COLON :

188 printf ("\t种 别 ： 冒 号\n");

189 printf ("\t位 置 : 第%d行\n" ,

华 中 科 技 大 学 课 程 实 验 报 告

tokenarray [i-1] .loc);

190 **break** ;

191 **case** SEMICON :

192 printf ("\t种 别 ： 分 号\n");

193 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

194 **break** ;

195 **case** ASSIGN :

196 printf ("\t种 别 ： 赋 值\n");

197 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

198 **break** ;

199 **case** IF:

200 printf ("\t种 别 ： if语 句\n");

201 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

202 **break** ;

203 **case** ELSE:

204 printf ("\t种 别 ： else语 句\n");

205 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

206 **break** ;

207 **case** SWITCH :

208 printf ("\t种 别 ： switch语 句\n");

209 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

210 **break** ;

211 **case** CASE:

212 printf ("\t种 别 ： case语 句\n");

213 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

214 **break** ;

215 **case** DEFAULT :

216 printf ("\t种 别 ： default语 句\n");

217 printf ("\t位 置 : 第%d行\n" ,

华 中 科 技 大 学 课 程 实 验 报 告

tokenarray [i-1] .loc);

218 **break** ;

219 **case** WHILE :

220 printf ("\t种 别 ： while语 句\n");

221 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

222 **break** ;

223 **case** DO:

224 printf ("\t种 别 ： do语 句\n");

225 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

226 **break** ;

227 **case** FOR:

228 printf ("\t种 别 ： for语 句\n");

229 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

230 **break** ;

231 **case** BREAK :

232 printf ("\t种 别 ： break语 句\n");

233 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

234 **break** ;

235 **case** CONTINUE :

236 printf ("\t种 别 ： continue语 句\n");

237 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

238 **break** ;

239 **case** RETURN :

240 printf ("\t种 别 ： return语 句\n");

241 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

242 **break** ;

243 **case** MAC:

244 printf ("\t种 别 ： 宏\n");

245 printf ("\t 内 容 ： %s\n" ,tokenarray [

华 中 科 技 大 学 课 程 实 验 报 告

i-1] .value);

246 printf ("\t位 置 : 第%d行\n" , tokenarray [i-1] .loc);

247 **break** ; 248 }

249 }

250 }

251 **void** lexerr (**int** a) 252 {

253 **switch** (a) 254 {

255 **case** STR\_NO\_R\_QUTION :

256 printf (" 第%d 行 ： 字 符 串 缺 少 右 双 引 号 。 \n" , lin);

257 **break** ;

258 **case** NUM\_BIN\_TYPE :

259 printf (" 第%d 行 ： 二 进 制 常 量 无 值 。 \n" ,lin);

260 **break** ;

261 **case** NUM\_HEX\_TYPE :

262 printf (" 第%d 行 ： 十 六 进 制 常 量 无 值 。 \n" , lin);

263 **break** ;

264 **case** CHAR\_NO\_R\_QUTION :

265 printf (" 第%d 行 ： 字 符 类 型 缺 少右 单 引 号 。 \n" ,lin);

266 **break** ;

267 **case** CHAR\_NO\_DATA :

268 printf (" 第%d 行 ： 字 符 常 量 无 值 。 \n" ,lin);

269 **break** ;

270 **case** OR\_NO\_PAIR :

271 printf (" 第%d 行 ： 逻 辑 或 缺 少 | 。\n" ,lin);

272 **break** ;

273 **case** TOKEN\_NO\_EXIST :

274 printf (" 第%d 行 ： 错 误 的 词 法 记 号 。 \n" ,lin);

275 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

276 }

277 error=1;

278 }

279 **#endif**

lexer\_head

1 **#ifndef** \_\_LEXER\_HEAD\_\_

2 **#define** \_\_LEXER\_HEAD\_\_

3 **void** wordscanner (FILE\*fp ,**char** file []);**//**读 取 文 件。

4 **int** nextstep (**char** file [],**int** cnt ,**char**\*present ,**char**\*next);**//**下 一 步。

5 **void** token\_print\_to\_file (FILE\*fp ,Token tokenarray []);**//**打 印 词 法 记 号 到 文 件 。

6 **void** token\_print\_to\_stdout (Token tokenarray []);**//**打 印 词 法 记 号 到 **stdout**。

7 **void** lexerr (**int** a);**//**处 理 词 法 记 号 错 误。 8

9 **const** **char**\*keywords []={"int" ,"char" ,"float" ,"double" ,"void" ," extern" ,"if" ,"else" ,"switch" ,"case" ,"default" ,"while" ,"do" ," for" ,"break" ,"continue" ,"return" ,NULL};**//** 关 键 字。

10 **int** lin=1,error=0; **//**行 号 和 错 误 标 记 11 **#endif**

lexer\_main

1 **#include**"lexer\_def .h"

2 **int** main ()

3 {**int** i=0,j=0,k=0,exi=0;

4 **char** present= '0 ' ;

5 **char** next= '0 ' ;

6 **char** file[MAXLEN];

7 Token tokenarray [MAXLEN];

8 FILE\*Token=fopen ("temp\_token .txt" ,"w");

9 FILE\*Code=fopen (" . ./input .txt" ,"r");

10 **if** (!Code) 11 {

12 printf ("文 件 打 开 失 败 。 ");

13 exit (-1);

14 }

15 **if** (!Token) 16 {

17 printf ("文 件 输 出 失 败 。 ");

18 exit (-1);

19 }

20 **//printf** **("**请 输 入 代 码 ， 以**^Z**结 尾 。 **\n")** **;**

21 wordscanner (Code ,file);

22 fclose (Code);

23 present=next=file [0];

24 **while** (nextstep (file ,i++ ,&present ,&next)!=0)**//** 等 于**0**则 文 件 结 束 ， 跳 出 循 环 。

25 {

26 **if** (present== ' ' | |present== ' '){**continue** ;}

27 **if** (present== '\n ') 28 {

29 lin++;

30 **continue** ; 31 }**//**清 除 空 格 和 回 车。

32 **else** **if** (present== '/ ')**//**清 除 注 释。 33 {

34 **if** (next== '/ ')**////** 型 注 释。 35 {

36 **while** (next != '\n '){nextstep (file ,i++ ,& present ,&next);}

37 nextstep (file ,i++ ,&present ,&next); 38 }

39 **else** **if** (next== '\* ')**///\*** 型 注 释。

40 {

41 **while** (1) 42 {

43 nextstep (file ,i++ ,&present ,&next)

;

44 **if** (present== '\* '&&next== '/ ')

45 {

46 nextstep (file ,i++ ,& present ,&next);

47 exi=1;**//** 注 释 结 束 ， 跳 出 循 环。

48 **break** ; 49 }

50 }

51 **if** (exi==1)**//** 注 释 结 束 ， 跳 出 循 环。 52 {

53 exi=0;

54 **continue** ; 55 }

56 }

57 **else**

58 {

59 tokenarray [j] .type=DIV;

60 tokenarray [j] .loc=lin;

61 j++; 62 }

63 }

64 **else** **if** (present== '# ')**//**判 断 宏 65 {

66 tokenarray [j] .type=MAC;

67 tokenarray [j] .loc=lin;

68 j++;

69 k=0;

70 **while** ((present== '\\ '&&next== '\n ') | |next != '\n ') 71 {

72 tokenarray [j-1] .value [k]=present ;

73 k++;

74 nextstep (file ,i++ ,&present ,&next); 75 }

76 tokenarray [j-1] .value [k]=present ;

77 k++;

华 中 科 技 大 学 课 程 实 验 报 告

78 tokenarray [j-1] .value [k]= '\0 ' ;

79 k=0; 80 }

81 **else** **if** (isalpha (present) !=0 | |present== '\_ ')**//**判 断 是 否 是 标 识 符 。

82 {

83 k=0;

84 tokenarray [j] .type=ID;

85 tokenarray [j] .loc=lin;

86 j++;

87 **while** (isalnum (next) !=0 | |next== '\_ ') 88 {

89 tokenarray [j-1] .value [k]=present ;

90 k++;

91 nextstep (file ,i++ ,&present ,&next); 92 }

93 tokenarray [j-1] .value [k]=present ;

94 k++;

95 tokenarray [j-1] .value [k]= '\0 ' ;

96 k=0;

97 **while** (keywords [k]!=NULL)**//**判 断 是 否 是 关 键 字。 98 {

99 k++;

100 **if**(strcmp (keywords [k-1] ,tokenarray [j-1] . value)==0)**//** 是 ， 把**type**设 置 成相 应 的 关 键 字。

101 {

102 **if**(k-1<=5){tokenarray [j-1] .type= INT+k-1;}

103 **else**{tokenarray [j-1] .type=IF+k -7;}

104 **break** ; 105 }

106 }

107 k=0;

华 中 科 技 大 学 课 程 实 验 报 告

108 **continue** ; 109 }

110 **else** **if**(isdigit (present)!=0)**//**判 断 是 否 是 数 字 常 量。 111 {

112 tokenarray [j] .type=NUM;

113 tokenarray [j] .loc=lin;

114 j++;

115 k=0;

116 **if** (present != '0 ' | |(present== '0 '&&next== ' . '))**//**判 断 是 否 是 否 是 非零 十 进 制数 字 常 量。

117 {

118 **while** (isdigit (next) !=0 | |next== ' . ') 119 {

120 tokenarray [j-1] .value [k++]= present ;

121 nextstep (file ,i++ ,&present ,&next)

;

122 }

123 tokenarray [j-1] .value [k]=present ;

124 k++;

125 tokenarray [j-1] .value [k]= '\0 ' ;

126 **continue** ; 127 }

128 **else** **if** (next<= '7 '&&next>= '0 ')**//**判 断 是 否 是 否 是 非 零 八 进 制数 字 常 量。

129 {

130 nextstep (file ,i++ ,&present ,&next);

131 **while** ((next<= '7 '&&next>= '0 ') | |next== ' . ') 132 {

133 tokenarray [j-1] .value [k]=present ;

134 k++;

135 nextstep (file ,i++ ,&present ,&next)

;

136 }

137 tokenarray [j-1] .value [k]=present ;

华 中 科 技 大 学 课 程 实 验 报 告

138 k++;

139 tokenarray [j-1] .value [k]= '\0 ' ;

140 **continue** ; 141 }

142 **else** **if** (next== 'b ')**//**判 断 是 否 是 否 是 非 零 二 进 制数 字 常 量 。

143 {

144 nextstep (file ,i++ ,&present ,&next);

145 **if** (next != '0 '&&next != '1 ') 146 {

147 tokenarray [j-1] .type=ERR;

148 lexerr (NUM\_BIN\_TYPE);

149 **continue** ; 150 }

151 nextstep (file ,i++ ,&present ,&next);

152 **while** (next== '0 ' | |next== '1 ' | |next== ' . ') 153 {

154 tokenarray [j-1] .value [k++]= present ;

155 nextstep (file ,i++ ,&present ,&next)

;

156 }

157 tokenarray [j-1] .value [k++]=present ;

158 tokenarray [j-1] .value [k]= '\0 ' ;

159 **continue** ; 160 }

161 **else** **if** (next== 'x ')**//**判 断 是 否 是 否 是 非 零 十 六 进 制数 字 常 量 。

162 {

163 nextstep (file ,i++ ,&present ,&next);

164 **if** (!((next<= 'F '&&next>= 'A ') | |(next>= 'a '&& next<= 'f ') | |isdigit (next)!=0))

165 {

166 tokenarray [j-1] .type=ERR;

167 lexerr (NUM\_HEX\_TYPE);

华 中 科 技 大 学 课 程 实 验 报 告

168 **continue** ; 169 }

170 nextstep (file ,i++ ,&present ,&next);

171 **while** ((next<= 'F '&&next>= 'A ') | |(next>= 'a ' &&next<= 'f ') | |isdigit (next) !=0 | |next== ' . ')

172 {

173 tokenarray [j-1] .value [k++]= present ;

174 nextstep (file ,i++ ,&present ,&next)

;

175 }

176 tokenarray [j-1] .value [k++]=present ;

177 tokenarray [j-1] .value [k]= '\0 ' ;

178 **continue** ; 179 }

180 **else//**判 断 是 否 是 零。 181 {

182 tokenarray [j-1] .value [0]= '0 ' ;

183 tokenarray [j-1] .value [1]= '\0 ' ;

184 **continue** ;

185 }

186 }

187 **else** **if** (present== '\' ')**//**判 断 是 否 是 字 符 常 量。 188 {

189 tokenarray [j] .type=CH;

190 tokenarray [j] .loc=lin;

191 j++;

192 **if** (next== '\' ') 193 {

194 tokenarray [j-1] .type=ERR;

195 lexerr (CHAR\_NO\_DATA);

196 **continue** ;

197 }

198 **else**

华 中 科 技 大 学 课 程 实 验 报 告

199 {

200 tokenarray [j-1] .value [0]=next;

201 tokenarray [j-1] .value [1]= '\0 ' ;

202 nextstep (file ,i++ ,&present ,&next);

203 nextstep (file ,i++ ,&present ,&next);

204 **if** (present != '\' ') 205 {

206 tokenarray [j-1] .type=ERR;

207 lexerr (CHAR\_NO\_R\_QUTION);

208 **while** (present != '\' ' | |present != ' ' || present != '\n '){nextstep (file ,i++ ,& present ,&next);}

209 }

210 **continue** ; 211 }

212 }

213 **else** **if** (present== '" ')**//**判 断 是 否 是 字 符 串 常 量。 214 {

215 tokenarray [j] .type=STR;

216 tokenarray [j] .loc=lin;

217 j++;

218 nextstep (file ,i++ ,&present ,&next);

219 k=0;

220 **while** (present != '" ') 221 {

222 tokenarray [j-1] .value [k++]=present ;

223 nextstep (file ,i++ ,&present ,&next);

224 **if** (present== '\n ')

225 {

226 tokenarray [j-1] .type=ERR;

227 lexerr (STR\_NO\_R\_QUTION);

228 **break** ; 229 }

230 }

231 tokenarray [j-1] .value [k]= '\0 ' ;

华 中 科 技 大 学 课 程 实 验 报 告

232 **continue** ; 233 }

234 **else//**判 断 分 隔 符 和 运 算 符。 235 {

236 **switch** (present) 237 {

238 **case** '+ ' :

239 tokenarray [j] .type=ADD;

240 tokenarray [j] .loc=lin;

241 j++;

242 **if** (next== '+ ')

243 {

244 tokenarray [j-1] .type=INC;

245 nextstep (file ,i++ ,& present ,&next);

246 }

247 **break** ;

248 **case** '- ' :

249 tokenarray [j] .type=SUB;

250 tokenarray [j] .loc=lin;

251 j++;

252 **if** (next== '- ') 253 {

254 tokenarray [j-1] .type=DEC;

255 nextstep (file ,i++ ,& present ,&next);

256 }

257 **break** ;

258 **case** '\* ' :

259 tokenarray [j] .type=MUL;

260 tokenarray [j] .loc=lin;

261 j++;

262 **break** ;

263 **case** '/ ' :

264 tokenarray [j] .type=DIV;

华 中 科 技 大 学 课 程 实 验 报 告

265 tokenarray [j] .loc=lin;

266 j++;

267 **break** ;

268 **case** '% ' :

269 tokenarray [j] .type=MOD;

270 tokenarray [j] .loc=lin;

271 j++;

272 **break** ;

273 **case** '> ' :

274 tokenarray [j] .type=GT;

275 tokenarray [j] .loc=lin;

276 j++;

277 **if** (next== '= ') 278 {

279 tokenarray [j-1] .type=GE;

280 nextstep (file ,i++ ,& present ,&next);

281 }

282 **break** ;

283 **case** '< ' :

284 tokenarray [j] .type=LT;

285 tokenarray [j] .loc=lin;

286 j++;

287 **if** (next== '- ') 288 {

289 tokenarray [j-1] .type=LE;

290 nextstep (file ,i++ ,& present ,&next);

291 }

292 **break** ;

293 **case** '= ' :

294 tokenarray [j] .type=ASSIGN ;

295 tokenarray [j] .loc=lin;

296 j++;

297 **if** (next== '= ')

华 中 科 技 大 学 课 程 实 验 报 告

298 {

299 tokenarray [j-1] .type=EQU;

300 nextstep (file ,i++ ,& present ,&next);

301 }

302 **break** ;

303 **case** '& ' :

304 tokenarray [j] .type=LEA;

305 tokenarray [j] .loc=lin;

306 j++;

307 **if** (next== '& ') 308 {

309 tokenarray [j-1] .type=AND;

310 nextstep (file ,i++ ,& present ,&next);

311 }

312 **break** ;

313 **case** ' | ' :

314 **if** (next== ' | ') 315 {

316 tokenarray [j] .type=OR;

317 tokenarray [j] .loc=lin;

318 j++;

319 nextstep (file ,i++ ,& present ,&next);

320 }

321 **else**

322 {

323 tokenarray [j++] .type=ERR;

324 lexerr (OR\_NO\_PAIR); 325 }

326 **break** ;

327 **case** ' ! ' :

328 tokenarray [j] .type=NOT;

329 tokenarray [j] .loc=lin;

华 中 科 技 大 学 课 程 实 验 报 告

330 j++;

331 **if** (next== '= ') 332 {

333 tokenarray [j-1] .type=NEQU

;

334 nextstep (file ,i++ ,& present ,&next);

335 }

336 **break** ;

337 **case** ' , ' :

338 tokenarray [j] .type=COMMA ;

339 tokenarray [j] .loc=lin;

340 j++;

341 **break** ;

342 **case** ' : ' :

343 tokenarray [j] .type=COLON ;

344 tokenarray [j] .loc=lin;

345 j++;

346 **break** ;

347 **case** ' ; ' :

348 tokenarray [j] .type=SEMICON ;

349 tokenarray [j] .loc=lin;

350 j++;

351 **break** ;

352 **case** '( ' :

353 tokenarray [j] .type=LPAREN ;

354 tokenarray [j] .loc=lin;

355 j++;

356 **break** ;

357 **case** ') ' :

358 tokenarray [j] .type=RPAREN ;

359 tokenarray [j] .loc=lin;

360 j++;

361 **break** ;

362 **case** ' [ ' :

华 中 科 技 大 学 课 程 实 验 报 告

363 tokenarray [j] .type=LBRACK ;

364 tokenarray [j] .loc=lin;

365 j++;

366 **break** ;

367 **case** '] ' :

368 tokenarray [j] .type=RBRACK ;

369 tokenarray [j] .loc=lin;

370 j++;

371 **break** ;

372 **case** '{ ' :

373 tokenarray [j] .type=LBRACE ;

374 tokenarray [j] .loc=lin;

375 j++;

376 **break** ;

377 **case** '} ' :

378 tokenarray [j] .type=RBRACE ;

379 tokenarray [j] .loc=lin;

380 j++;

381 **break** ;

382 **default** :

383 tokenarray [j++] .type=ERR;

384 lexerr (TOKEN\_NO\_EXIST);

385 **break** ;

386 }

387 }

388 }

389 tokenarray [j] .type=END;

390 tokenarray [j] .loc=lin;

391 token\_print\_to\_stdout (tokenarray);

392 token\_print\_to\_file (Token ,tokenarray);

393 fclose (Token);

394 **return** 0; 395 }

**9.3** 语法分析器(**parser**)

parser\_datastructure

1 **#ifndef** \_\_PARSER\_DATASTRUCTURE\_\_

2 **#define** \_\_PARSER\_DATASTRUCTURE\_\_

3 **typedef** **struct** TOKEN**//**定 义**token** **.**

4 {

5 **int** type;

6 **char** value [MAXLEN\_OF\_CONTENT];

7 **int** loc; 8 }Token ;

9 **enum** Tag**//**词 法 记 号 标 签。

10 {

11 ERR ,**//**错 误。

12 END ,**//**文 件 结 束。

13 ID ,**//**标 识 符。

14 INT ,CHAR ,FLOAT ,DOUBLE ,VOID ,EXTERN ,**//**数 据 类 型。

15 NUM ,**//**数 字 常 量。

16 CH ,**//** 字 符 常 量。

17 STR ,**//** 字 符 串。

18 NOT ,LEA ,**//**！**&**

19 ADD ,SUB ,MUL ,DIV ,MOD ,**//+-\*/%**

20 INC ,DEC ,**//++--**

21 GT ,GE ,LT ,LE ,EQU ,NEQU ,**//>>=<<==!=**

22 AND ,OR ,**//&&** **|** **|**

23 LPAREN ,RPAREN ,**//** ()

24 LBRACK ,RBRACK ,**//**【 】

25 LBRACE ,RBRACE ,**//** {}

26 COMMA ,COLON ,SEMICON ,**//** ， ： ；

27 ASSIGN ,**//=**

28 IF ,ELSE ,SWITCH ,CASE ,DEFAULT ,WHILE ,DO ,FOR ,BREAK ,CONTINUE , RETURN ,**//** 关 键 字。

29 MAC**//** 宏。

30 };

31 **enum** synerror**///**语 法 错 误 标 签。

32 {

33 TYPELOST ,TYPEWRONG ,**//**类 型

34 IDLOST ,IDWRONG ,**//**标 识 符

35 NUMLOST ,NUMWRONG ,**//**数 组 的 长 度

36 LITERALLOST ,LITERALWRONG ,**//** 常 量

37 COMMALOST ,COMMAWRONG ,**//** **,**

38 SEMICONLOST ,SEMICONWRONG ,**//** **;**

39 ASSIGNLOST ,ASSIGNWRONG ,**//=**

40 COLONLOST ,COLONWRONG ,**//** **:**

41 STATEMENTLOST ,STATEMENTWRONG ,**//**语 句

42 LPARENLOST ,LPARENWRONG ,**//** **(**

43 RPARENLOST ,RPARENWRONG ,**//)**

44 LBRACKLOST ,LBRACKWRONG ,**//** **[**

45 RBRACKLOST ,PBRACKWRONG ,**//]**

46 LBRACELOST ,LBRACEWRONG ,**//{**

47 RBRACELOST ,RBRACEWRONG**//}**

48 };

49 **#endif**

parser\_def

1 **#ifndef** \_\_PARSER\_DEF\_\_

2 **#define** \_\_PARSER\_DEF\_\_

3 **#include**<stdio .h>

4 **#include**<stdlib .h>

5 **#include**<string .h>

6 **#include**<stdarg .h>

7 **#define** LOST 1

8 **#define** WRONG 0

9 **#define** MAXLEN\_OF\_CONTENT 50

10 **#define** MAXLEN 1000

11

12 **#include**"parser\_datastructure .h"

13 **#include**"parser\_head .h"

14 **#include**"parser\_func .c"

15 **#endif**

parser\_func

1 **#ifndef** \_\_PARSER\_FUNC\_\_

2 **#define** \_\_PARSER\_FUNC\_\_

3 **//\*\*\*\*\*\*\*\*\*\*\***控 制 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

4 **void** readtokenarray (FILE\*fp) 5 {

6 **int** i=0;

7 **do** 8 {

9 fscanf (fp ,"%d" ,&tokenarray [i] .type);

10 **if** (tokenarray [i] .type==ID | |tokenarray [i] .type== NUM | |tokenarray [i] .type==CH | |tokenarray [i] . type==STR | |tokenarray [i] .type==MAC)

11 {

12 fgetc (fp);**//** 去 除**\n** 字 符。

13 **//fscanf** **(fp** **,"%s",tokenarray** **[i]** **.value)** **;**

14 fgets (tokenarray [i] .value ,MAXLEN\_OF\_CONTENT , fp);

15 tokenarray [i] .value [strlen (tokenarray [i] . value)-1]= '\0 ' ;**//**删 除 末 尾 的 空 格。

16 }

17 fscanf (fp ,"%d" ,&tokenarray [i] .loc);

18 i++; 19 }

20 **while** (tokenarray [i-1] .type !=END); 21 }

22 **void** gonext ()

23 {

24 nexttype=tokenarray [noft] .type;

25 loc=tokenarray [noft] .loc;

26 noft++;

27 }

28 **void** in () 29 {

30 layer++;

31 cnt++;

32 }**//**维 护**AST** 的 层 数。

33 **void** out (){layer--;}

34 **void** block\_in (){blocknum++;}**//**维 护 所 处 块 的 层 数。

35 **void** block\_out (){blocknum--;}

36 **void** print\_blank ()**//**打 印**AST** 中 的 空 格。 37 {

38 **int** i=0;

39 **for**(i=0;i<layer ;i++){printf (" ");} 40 }

41 **void** fprint\_token (FILE\*fp) 42 {

43 **int** i=0;

44 **/\*if(noft-2>=0&&tokenarray** **[noft-2]** **.type==SEMICON)**

45 **{for(i=0;i<blocknum-1;i++){fprintf** **(fp** **,"** **")** **;}}**

46 **if(noft-2>=0&&tokenarray** **[noft-2]** **.type==LBRACE)** 47 **{for(i=0;i<blocknum-1;i++){fprintf** **(fp** **,"** **")** **;}}\*/**

48 **switch** (tokenarray [noft-1] .type) 49 {

50 **case** ID:

51 fprintf (fp ,"%s " ,tokenarray [noft-1] .value );

52 **break** ;

53 **case** INT:

54 fprintf (fp ,"int ");

55 **break** ;

56 **case** CHAR:

57 fprintf (fp ,"char ");

58 **break** ;

59 **case** FLOAT :

60 fprintf (fp ,"float ");

61 **break** ;

62 **case** DOUBLE :

63 fprintf (fp ,"double ");

64 **break** ;

65 **case** VOID:

66 fprintf (fp ,"void ");

67 **break** ;

68 **case** EXTERN :

69 fprintf (fp ,"extern ");

70 **break** ;

71 **case** NUM:

72 fprintf (fp ,"%s " ,tokenarray [noft-1] .value );

73 **break** ;

74 **case** CH:

75 fprintf (fp ,"\'%s\' " ,tokenarray [noft-1] . value);

76 **break** ;

77 **case** STR:

78 fprintf (fp ,"\"%s\" " ,tokenarray [noft-1] . value);

79 **break** ;

80 **case** NOT:

81 fprintf (fp ," ! ");

82 **break** ;

83 **case** LEA:

84 fprintf (fp ,"& ");

85 **break** ;

86 **case** ADD:

87 fprintf (fp ,"+ ");

88 **break** ;

89 **case** SUB:

90 fprintf (fp ,"- ");

91 **break** ;

92 **case** MUL:

93 fprintf (fp ,"\* ");

94 **break** ;

95 **case** DIV:

96 fprintf (fp ,"/ ");

华 中 科 技 大 学 课 程 实 验 报 告

97 **break** ;

98 **case** MOD:

99 fprintf (fp ,"%% ");

100 **break** ;

101 **case** INC:

102 fprintf (fp ,"++ ");

103 **break** ;

104 **case** DEC:

105 fprintf (fp ,"-- ");

106 **break** ;

107 **case** GT:

108 fprintf (fp ,"> ");

109 **break** ;

110 **case** GE:

111 fprintf (fp ,">= ");

112 **break** ;

113 **case** LT:

114 fprintf (fp ,"< ");

115 **break** ;

116 **case** LE:

117 fprintf (fp ,"<= ");

118 **break** ;

119 **case** EQU:

120 fprintf (fp ,"== ");

121 **break** ;

122 **case** NEQU:

123 fprintf (fp ," != ");

124 **break** ;

125 **case** AND:

126 fprintf (fp ,"&& ");

127 **break** ;

128 **case** OR:

129 fprintf (fp ," || ");

130 **break** ;

131 **case** LPAREN :

华 中 科 技 大 学 课 程 实 验 报 告

132 fprintf (fp ," ( ");

133 **break** ;

134 **case** RPAREN :

135 fprintf (fp ,") ");

136 **if** (tokenarray [noft] .type !=RBRACE&& tokenarray [noft] .type !=SEMICON&& tokenarray [noft] .type !=ADD&&tokenarray [noft] .type !=SUB&&tokenarray [noft] . type !=MUL&&tokenarray [noft] .type !=MOD &&tokenarray [noft] .type !=DIV)

137 {

138 fprintf (fp ,"\n");

139 **for**(i=0;i<1+blocknum ;i++){fprintf (fp ," ");}

140 }

141 **break** ;

142 **case** LBRACK :

143 fprintf (fp ," [ ");

144 **break** ;

145 **case** RBRACK :

146 fprintf (fp ,"] ");

147 **break** ;

148 **case** LBRACE :

149 fprintf (fp ,"\n");

150 **for**(i=0;i<blocknum-1;i++){fprintf (fp ," " );}

151 fprintf (fp ,"{\n");

152 **for**(i=0;i<blocknum ;i++){fprintf (fp ," " );}

153 **break** ;

154 **case** RBRACE :

155 **for**(i=0;i<blocknum-1;i++){fprintf (fp ," " );}

156 fprintf (fp ,"}\n");

157 **if** (tokenarray [noft] .type !=RBRACE){**for**(i

华 中 科 技 大 学 课 程 实 验 报 告

=0;i<blocknum-1;i++){fprintf (fp ," ");}}

158 **break** ;

159 **case** COMMA :

160 fprintf (fp ," , ");

161 **break** ;

162 **case** COLON :

163 fprintf (fp ," :\n");

164 **for**(i=0;i<blocknum ;i++){fprintf (fp ," " );}

165 fprintf (fp ," ");

166 **break** ;

167 **case** SEMICON :

168 fprintf (fp ," ;");

169 **if** (isinfor !=1) 170 {

171 fprintf (fp ,"\n");

172 **if** (if2nd==1) 173 {

174 **if** (tokenarray [noft] .type !=RBRACE){**for**(i=0;i<

blocknum-1;i++){

fprintf (fp ," ")

;}}

175 if2nd=0; 176 }

177 **else** 178 {

179 **if** (tokenarray [noft] .type !=RBRACE){**for**(i=0;i< blocknum ;i++){fprintf ( fp ," ");}}

180 }

181 }

182 **else**{fprintf (fp ," ");}

华 中 科 技 大 学 课 程 实 验 报 告

183 **break** ;

184 **case** ASSIGN :

185 fprintf (fp ,"= ");

186 **break** ;

187 **case** IF:

188 fprintf (fp ,"if ");

189 **break** ;

190 **case** ELSE:

191 fprintf (fp ,"else ");

192 **if** (tokenarray [noft] .type !=RBRACE) 193 {

194 fprintf (fp ,"\n");

195 **for**(i=0;i<1+blocknum ;i++){fprintf (fp ," ");}

196 }

197 **break** ;

198 **case** SWITCH :

199 fprintf (fp ,"switch ");

200 **break** ;

201 **case** CASE:

202 fprintf (fp ,"case ");

203 **break** ;

204 **case** DEFAULT :

205 fprintf (fp ,"default ");

206 **break** ;

207 **case** WHILE :

208 fprintf (fp ,"while ");

209 **break** ;

210 **case** DO:

211 fprintf (fp ,"do ");

212 **break** ;

213 **case** FOR:

214 fprintf (fp ,"for ");

215 **break** ;

216 **case** BREAK :

华 中 科 技 大 学 课 程 实 验 报 告

217 fprintf (fp ,"break ");

218 **break** ;

219 **case** CONTINUE :

220 fprintf (fp ,"continue ");

221 **break** ;

222 **case** RETURN :

223 fprintf (fp ,"return ");

224 **break** ;

225 **case** MAC:

226 fprintf (fp ,"%s\n " ,tokenarray [noft-1] . value);

227 **for**(i=0;i<blocknum ;i++){fprintf (fp ," " );}

228 **break** ;

229 **default** :

230 printf ("无 法 识 别 的 词 法 记 号！ \n"); 231 }

232 }

233 **//\*\*\*\*\*\*\*\*\*\*\***绘 制**AST** 的 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

234 **void** init\_ast () 235 {

236 graph=fopen (" . ./AST .dot" ,"w");

237 fprintf (graph ,"digraph AST {\n");

238 fprintf (graph ,"node [shape=ellipse , fontname=\"KaiTi\"];\ n");

239 fprintf (graph ,"{\n"); 240 }

241 **void** end\_ast ()

242 {

243 fprintf (graph ,"}\n");

244 fprintf (graph ,"}\n");

245 fclose (graph);

246 }

247 **void** print\_ast (**int** from ,**const** **char**\*from\_name ,**int** to ,**const** **char**\* to\_name ,**int** is\_terminal)

华 中 科 技 大 学 课 程 实 验 报 告

248 {

249 **if** (setted [from]==0)**//**未 设 置 250 {

251 setted [from]=1;

252 **if**(is\_terminal==1)**//**是 终 结 符 253 {

254 fprintf (graph ,"%d[label=\"%s\",shape=box ];\n" ,from ,from\_name);

255 }

256 **else** 257 {

258 fprintf (graph ,"%d [label=\"%s\"];\n" ,from , from\_name);

259 }

260 }

261 **if** (setted [to]==0) 262 {

263 setted [to]=1;

264 **if** (is\_terminal==1) 265 {

266 fprintf (graph ,"%d[label=\"%s\",shape=box ];\n" ,to ,to\_name);

267 }

268 **else** 269 {

270 fprintf (graph ,"%d [label=\"%s\"];\n" ,to , to\_name);

271 }

272 }

273 fprintf (graph ,"%d->%d;\n" ,from ,to);

274 }

275 **//\*\*\*\*\*\*\*\*\*\*** 变 量 声 明 函 数 **\*\*\*\*\*\*\*\*\*\*\*//**

276 **void** program () 277 {

278 **int** from=cnt;

华 中 科 技 大 学 课 程 实 验 报 告

279 **if** (nexttype==END) 280 {

281 **return** ; 282 }

283 cnt++;

284 print\_ast (from ,"program" ,cnt ,"segment" ,0);

285 segment ();

286 cnt++;

287 print\_ast (from ,"program" ,cnt ,"program" ,0);

288 program ();

289 }

290 **void** segment () 291 {

292 **int** from=cnt;

293 **if** (nexttype==MAC) 294 {

295 cnt++;

296 print\_ast (from ,"segment" ,cnt ,tokenarray [noft-1] . value ,1);

297 fprint\_token (output);

298 printf (" 宏 ： %s\n" ,tokenarray [noft-1] .value);

299 gonext ();

300 }

301 **else** **if** (nexttype==EXTERN) 302 {

303 cnt++;

304 print\_ast (from ,"segment" ,cnt ,"extern" ,1);

305 fprint\_token (output);

306 printf ("外 部 变 量 :\n");

307 gonext ();

308 }

309 **else** 310 {

311 printf ("变 量 声 明 ： \n");

312 in ();

华 中 科 技 大 学 课 程 实 验 报 告

313 print\_ast (from ,"segment" ,cnt ,"type" ,0);

314 type ();

315 out ();

316 in ();

317 print\_ast (from ,"segment" ,cnt ,"def" ,0);

318 def ();

319 out (); 320 }

321 }

322 **void** type () 323 {

324 **int** from=cnt;

325 print\_blank ();

326 **switch** (nexttype) 327 {

328 **case** INT:

329 cnt++;

330 print\_ast (from ,"type" ,cnt ,"int" ,1);

331 fprint\_token (output);

332 printf ("类 型 ： int\n");

333 gonext ();

334 **break** ;

335 **case** CHAR:

336 cnt++;

337 print\_ast (from ,"type" ,cnt ,"char" ,1);

338 fprint\_token (output);

339 printf ("类 型 ： char\n");

340 gonext ();

341 **break** ;

342 **case** FLOAT :

343 cnt++;

344 print\_ast (from ,"type" ,cnt ,"float" ,1);

345 fprint\_token (output);

346 printf ("类 型 ： float\n");

347 gonext ();

华 中 科 技 大 学 课 程 实 验 报 告

348 **break** ;

349 **case** DOUBLE :

350 cnt++;

351 print\_ast (from ,"type" ,cnt ,"double" ,1);

352 fprint\_token (output);

353 printf ("类 型 ： double\n");

354 gonext ();

355 **break** ;

356 **case** VOID:

357 cnt++;

358 print\_ast (from ,"type" ,cnt ,"void" ,1);

359 fprint\_token (output);

360 printf ("类 型 ： void\n");

361 gonext ();

362 **break** ;

363 **default** :

364 synerr (nexttype ,3,INT ,CHAR ,FLOAT ,DOUBLE , VOID);

365 **break** ; 366 }

367 }

368 **void** def () 369 {

370 **int** from=cnt;

371 **if** (nexttype==MUL) 372 {

373 cnt++;

374 print\_ast (from ,"def" ,cnt ,"\*" ,1);

375 fprint\_token (output);

376 print\_blank ();

377 printf ("指 针\n");

378 gonext ();

379 **if** (nexttype==ID)

380 {

381 cnt++;

华 中 科 技 大 学 课 程 实 验 报 告

382 print\_ast (from ,"def" ,cnt ,tokenarray [noft -1] .value ,1);

383 fprint\_token (output);

384 print\_blank ();

385 printf ("名 称 ： %s\n" ,tokenarray [noft-1] . value);

386 gonext ();

387 in ();

388 print\_ast (from ,"def" ,cnt ,"init" ,0);

389 init ();

390 out ();

391 in ();

392 print\_ast (from ,"def" ,cnt ,"deflist" ,0);

393 deflist ();

394 out ();

395 }

396 **else** 397 {

398 synerr (nexttype ,1,ID); 399 }

400 }

401 **else** **if** (nexttype==ID) 402 {

403 cnt++;

404 print\_ast (from ,"def" ,cnt ,tokenarray [noft-1] .value ,1);

405 fprint\_token (output);

406 print\_blank ();

407 printf ("名 称 ： %s\n" ,tokenarray [noft-1] .value);

408 gonext ();

409 in ();

410 print\_ast (from ,"def" ,cnt ,"idtail" ,0);

411 idtail ();

412 out (); 413 }

华 中 科 技 大 学 课 程 实 验 报 告

414 **else** 415 {

416 synerr (nexttype ,1,ID); 417 }

418 }

419 **void** init () 420 {

421 **int** from=cnt;

422 **if** (nexttype==ASSIGN) 423 {

424 cnt++;

425 print\_ast (from ,"init" ,cnt ,"=" ,1);

426 fprint\_token (output);

427 print\_blank ();

428 printf ("赋 值\n");

429 gonext ();

430 in ();

431 print\_ast (from ,"init" ,cnt ,"expr" ,0);

432 expr ();

433 out ();

434 }

435 }

436 **void** deflist () 437 {

438 **int** from=cnt;

439 **if** (nexttype==COMMA) 440 {

441 cnt++;

442 print\_ast (from ,"deflist" ,cnt ," ," ,1);

443 fprint\_token (output);

444 gonext ();

445 in ();

446 print\_ast (from ,"deflist" ,cnt ,"defdata" ,0);

447 defdata ();

448 out ();

华 中 科 技 大 学 课 程 实 验 报 告

449 in ();

450 print\_ast (from ,"deflist" ,cnt ,"deflist" ,0);

451 deflist ();

452 out (); 453 }

454 **else** **if** (nexttype==SEMICON) 455 {

456 cnt++;

457 print\_ast (from ,"deflist" ,cnt ," ;" ,1);

458 fprint\_token (output);

459 gonext ();

460 }

461 **else** 462 {

463 synerr (nexttype ,2,COMMA ,SEMICON); 464 }

465 }

466 **void** idtail () 467 {

468 **int** from=cnt;

469 **if** (nexttype==LPAREN) 470 {

471 cnt++;

472 print\_ast (from ,"idtail" ,cnt ," (" ,1);

473 fprint\_token (output);

474 gonext ();

475 in ();

476 print\_ast (from ,"idtail" ,cnt ,"para" ,0);

477 para ();

478 out ();

479 **if** (nexttype==RPAREN)

480 {

481 cnt++;

482 print\_ast (from ,"idtail" ,cnt ,")" ,1);

483 fprint\_token (output);

华 中 科 技 大 学 课 程 实 验 报 告

484 gonext ();

485 in ();

486 print\_ast (from ,"idtail" ,cnt ,"funtail" ,0);

487 funtail ();

488 out (); 489 }

490 **else** 491 {

492 synerr (nexttype ,1,RPAREN); 493 }

494 }

495 **else**

496 {

497 in ();

498 print\_ast (from ,"idtail" ,cnt ,"varrdef" ,0);

499 varrdef ();

500 out ();

501 in ();

502 print\_ast (from ,"idtail" ,cnt ,"deflist" ,0);

503 deflist ();

504 out (); 505 }

506 }

507 **void** defdata () 508 {

509 **int** from=cnt;

510 **if** (nexttype==MUL) 511 {

512 cnt++;

513 print\_ast (from ,"defdata" ,cnt ,"\*" ,1);

514 fprint\_token (output);

515 print\_blank ();

516 printf ("指 针\n");

517 gonext ();

518 **if** (nexttype==ID)

华 中 科 技 大 学 课 程 实 验 报 告

519 {

520 cnt++;

521 print\_ast (from ,"defdata" ,cnt ,tokenarray [ noft-1] .value ,1);

522 fprint\_token (output);

523 print\_blank ();

524 printf ("名 称 ： %s\n" ,tokenarray [noft-1] . value);

525 gonext ();

526 in ();

527 print\_ast (from ,"defdata" ,cnt ,"init" ,0);

528 init ();

529 out (); 530 }

531 **else** 532 {

533 synerr (nexttype ,1,ID); 534 }

535 }

536 **else** **if** (nexttype==ID) 537 {

538 cnt++;

539 print\_ast (from ,"defdata" ,cnt ,tokenarray [noft-1] . value ,1);

540 fprint\_token (output);

541 print\_blank ();

542 printf ("名 称 ： %s\n" ,tokenarray [noft-1] .value);

543 gonext ();

544 in ();

545 print\_ast (from ,"defdata" ,cnt ,"varrdef" ,0);

546 varrdef ();

547 out (); 548 }

549 **else**

550 {

华 中 科 技 大 学 课 程 实 验 报 告

551 synerr (nexttype ,2,ID ,MUL);

552 }

553 }

554 **void** varrdef () 555 {

556 **int** from=cnt;

557 **if** (nexttype==LBRACK) 558 {

559 cnt++;

560 print\_ast (from ,"varrdef" ,cnt ," [" ,1);

561 fprint\_token (output);

562 gonext ();

563 **if** (nexttype==NUM)

564 {

565 cnt++;

566 print\_ast (from ,"varrdef" ,cnt ,tokenarray [ noft-1] .value ,1);

567 fprint\_token (output);

568 print\_blank ();

569 printf ("数 组 下 标 ： %s\n" ,tokenarray [noft -1] .value);

570 gonext ();

571 **if** (nexttype==RBRACK) 572 {

573 cnt++;

574 print\_ast (from ,"varrdef" ,cnt ,"]" ,1);

575 fprint\_token (output);

576 gonext ();

577 }

578 **else** 579 {

580 synerr (nexttype ,1,RBRACK); 581 }

582 }

华 中 科 技 大 学 课 程 实 验 报 告

583 **else** 584 {

585 synerr (nexttype ,1,NUM); 586 }

587 }

588 **else**

589 {

590 in ();

591 print\_ast (from ,"varrdef" ,cnt ,"init" ,0);

592 init ();

593 out (); 594 }

595 }

596 **//\*\*\*\*\*\*\*\*\*** 函 数 声 明 函 数 **\*\*\*\*\*\*\*\*\*//**

597 **void** funtail () 598 {

599 **int** from=cnt;

600 **if** (nexttype==SEMICON) 601 {

602 cnt++;

603 print\_ast (from ,"funtail" ,cnt ," ;" ,1);

604 fprint\_token (output);

605 gonext ();

606 }

607 **else**

608 {

609 print\_blank ();

610 printf ("作 用 域 内 容 ： \n");

611 block\_in ();

612 in ();

613 print\_ast (from ,"funtail" ,cnt ,"block" ,0);

614 block ();

615 out ();

616 block\_out (); 617 }

华 中 科 技 大 学 课 程 实 验 报 告

618 }

619 **void** para () 620 {

621 **int** from=cnt;

622 print\_blank ();

623 printf (" 函 数 形 参 ： \n");

624 **if** (nexttype==INT | |nexttype==DOUBLE | |nexttype==FLOAT | | nexttype==VOID | |nexttype==CHAR)

625 {

626 in ();

627 print\_ast (from ,"para" ,cnt ,"type" ,0);

628 type ();

629 out ();

630 in ();

631 print\_ast (from ,"para" ,cnt ,"paradata" ,0);

632 paradata ();

633 out ();

634 in ();

635 print\_ast (from ,"para" ,cnt ,"paralist" ,0);

636 paralist ();

637 out (); 638 }

639 }

640 **void** paradata () 641 {

642 **int** from=cnt;

643 **if** (nexttype==MUL) 644 {

645 cnt++;

646 print\_ast (from ,"paradata" ,cnt ,"\*" ,1);

647 fprint\_token (output);

648 print\_blank ();

649 printf ("指 针\n");

650 gonext ();

651 **if** (nexttype==ID)

华 中 科 技 大 学 课 程 实 验 报 告

652 {

653 cnt++;

654 print\_ast (from ,"paradata" ,cnt ,tokenarray [ noft-1] .value ,1);

655 fprint\_token (output);

656 print\_blank ();

657 printf ("名 称 ： %s\n" ,tokenarray [noft-1] . value);

658 gonext (); 659 }

660 **else** 661 {

662 synerr (nexttype ,1,ID); 663 }

664 }

665 **else** **if** (nexttype==ID) 666 {

667 cnt++;

668 print\_ast (from ,"paradata" ,cnt ,tokenarray [noft-1] . value ,1);

669 fprint\_token (output);

670 print\_blank ();

671 printf ("名 称 ： %s\n" ,tokenarray [noft-1] .value);

672 gonext ();

673 in ();

674 paradatatail ();

675 out ();

676 }

677 **else** 678 {

679 synerr (nexttype ,2,ID ,MUL); 680 }

681 }

682 **void** paralist ()

683 {

华 中 科 技 大 学 课 程 实 验 报 告

684 **int** from=cnt;

685 **if** (nexttype==COMMA) 686 {

687 cnt++;

688 print\_ast (from ,"paralist" ,cnt ," ," ,1);

689 fprint\_token (output);

690 gonext ();

691 in ();

692 print\_ast (from ,"paralist" ,cnt ,"type" ,0);

693 type ();

694 out ();

695 in ();

696 print\_ast (from ,"paralist" ,cnt ,"paradata" ,0);

697 paradata ();

698 out ();

699 in ();

700 print\_ast (from ,"paralist" ,cnt ,"paralist" ,0);

701 paralist ();

702 out ();

703 }

704 }

705 **void** paradatatail () 706 {

707 **int** from=cnt;

708 **if** (nexttype==LBRACK) 709 {

710 cnt++;

711 print\_ast (from ,"paradatatail" ,cnt ," [" ,1);

712 fprint\_token (output);

713 gonext ();

714 **if** (nexttype==NUM)

715 {

716 cnt++;

717 print\_ast (from ,"paradatatail" ,cnt , tokenarray [noft-1] .value ,1);

华 中 科 技 大 学 课 程 实 验 报 告

718 fprint\_token (output);

719 print\_blank ();

720 printf ("数 组 下 标 ： %s\n" ,tokenarray [noft -1] .value);

721 gonext ();

722 **if** (nexttype==RBRACK) 723 {

724 cnt++;

725 print\_ast (from ,"paradatatail" ,cnt ,"]" ,1);

726 fprint\_token (output);

727 gonext ();

728 }

729 **else** 730 {

731 synerr (nexttype ,1,RBRACK); 732 }

733 }

734 **else** 735 {

736 synerr (nexttype ,1,NUM); 737 }

738 }

739 }

740 **//\*\*\*\*\*\*\*\*\*\*\*\*\*** 函 数 定 义 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

741 **void** block () 742 {

743 **int** from=cnt;

744 **if** (nexttype==LBRACE) 745 {

746 cnt++;

747 print\_ast (from ,"block" ,cnt ,"{" ,1);

748 fprint\_token (output);

749 gonext ();

750 in ();

华 中 科 技 大 学 课 程 实 验 报 告

751 print\_ast (from ,"block" ,cnt ,"subprogram" ,0);

752 subprogram ();

753 out ();

754 **if** (nexttype==RBRACE) 755 {

756 cnt++;

757 print\_ast (from ,"block" ,cnt ,"}" ,1);

758 fprint\_token (output);

759 gonext ();

760 }

761 **else** 762 {

763 synerr (nexttype ,1,RBRACE); 764 }

765 }

766 **else** 767 {

768 synerr (nexttype ,1,LBRACE); 769 }

770 }

771 **void** subprogram () 772 {

773 **int** from=cnt;

774 **if** (nexttype==INT | |nexttype==DOUBLE | |nexttype==FLOAT | | nexttype==CHAR | |nexttype==VOID)**//**进 入 变 量 声 明。

775 {

776 in ();

777 cnt++;

778 print\_ast (from ,"subprogram" ,cnt ,"localdef" ,0);

779 localdef ();

780 out ();

781 in ();

782 print\_ast (from ,"subprogram" ,cnt ,"subprogram" ,0);

783 subprogram ();

784 out ();

华 中 科 技 大 学 课 程 实 验 报 告

785 }

786 **//else** **if** **(nexttype==ID** **||nexttype==MUL** **|** **|(nexttype<=RETURN**

**&&nexttype>=IF))//**进 入 语 句。

787 **else** **if** (nexttype==ID | |nexttype==MUL | |(nexttype<=RETURN&& nexttype>=IF&&nexttype !=CASE&&nexttype !=DEFAULT))**//**进 入 语 句 。

788 {

789 in ();

790 print\_ast (from ,"subprogram" ,cnt ,"statement" ,0);

791 statement ();

792 out ();

793 in ();

794 print\_ast (from ,"subprogram" ,cnt ,"subprogram" ,0);

795 subprogram ();

796 out (); 797 }

798 }

799 **void** localdef () 800 {

801 **int** from=cnt;

802 print\_blank ();

803 printf (" 自 动 变 量 ： \n");

804 in ();

805 print\_ast (from ,"localdef" ,cnt ,"type" ,0);

806 type ();

807 out ();

808 in ();

809 print\_ast (from ,"localdef" ,cnt ,"defdata" ,0);

810 defdata ();

811 out ();

812 in ();

813 print\_ast (from ,"localdef" ,cnt ,"deflist" ,0);

814 deflist ();

815 out ();

816 }

华 中 科 技 大 学 课 程 实 验 报 告

817 **//\*\*\*\*\*\*\*\*\*\*\*\***语 句 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*\*\*//**

818 **void** statement () 819 {

820 **int** from=cnt;

821 **switch** (nexttype) 822 {

823 **case** WHILE :

824 cnt++;

825 print\_ast (from ,"statement" ,cnt ,"while" ,1)

;

826 fprint\_token (output);

827 print\_blank ();

828 printf ("while语 句 :\n");

829 gonext ();

830 in ();

831 print\_ast (from ,"statement" ,cnt ,"whilestat " ,0);

832 whilestat ();

833 out ();

834 **break** ;

835 **case** FOR:

836 cnt++;

837 print\_ast (from ,"statement" ,cnt ,"for" ,1);

838 fprint\_token (output);

839 print\_blank ();

840 printf ("for语 句 :\n");

841 gonext ();

842 in ();

843 print\_ast (from ,"statement" ,cnt ,"forstat" ,0);

844 forstat ();

845 out ();

846 **break** ;

847 **case** DO:

848 cnt++;

华 中 科 技 大 学 课 程 实 验 报 告

849 print\_ast (from ,"statement" ,cnt ,"do" ,1);

850 fprint\_token (output);

851 print\_blank ();

852 printf ("do-while语 句 :\n");

853 gonext ();

854 in ();

855 print\_ast (from ,"statement" ,cnt ," dowhilestat" ,0);

856 dowhilestat ();

857 out ();

858 **break** ;

859 **case** IF:

860 cnt++;

861 print\_ast (from ,"statement" ,cnt ,"if" ,1);

862 fprint\_token (output);

863 print\_blank ();

864 printf ("if语 句 :\n");

865 gonext ();

866 in ();

867 print\_ast (from ,"statement" ,cnt ,"ifstat" ,0);

868 ifstat ();

869 out ();

870 **break** ;

871 **case** SWITCH :

872 cnt++;

873 print\_ast (from ,"statement" ,cnt ,"switch" ,1);

874 fprint\_token (output);

875 print\_blank ();

876 printf ("switch语 句 :\n");

877 gonext ();

878 in ();

879 print\_ast (from ,"statement" ,cnt ," switchstat" ,0);

华 中 科 技 大 学 课 程 实 验 报 告

880 switchstat ();

881 out ();

882 **break** ;

883 **case** BREAK :

884 cnt++;

885 print\_ast (from ,"statement" ,cnt ,"break" ,1)

;

886 fprint\_token (output);

887 print\_blank ();

888 printf ("break语 句 :\n");

889 gonext ();

890 **if** (nexttype==SEMICON) 891 {

892 cnt++;

893 print\_ast (from ,"statement" ,cnt ," ; " ,1);

894 fprint\_token (output);

895 gonext ();

896 }

897 **else** 898 {

899 synerr (nexttype ,1,SEMICON); 900 }

901 **break** ;

902 **case** CONTINUE :

903 cnt++;

904 print\_ast (from ,"statement" ,cnt ,"continue" ,1);

905 fprint\_token (output);

906 print\_blank ();

907 printf ("continue语 句 :\n");

908 gonext ();

909 **if** (nexttype==SEMICON)

910 {

911 cnt++;

华 中 科 技 大 学 课 程 实 验 报 告

912 print\_ast (from ,"statement" ,cnt ," ; " ,1);

913 fprint\_token (output);

914 gonext (); 915 }

916 **else** 917 {

918 synerr (nexttype ,1,SEMICON); 919 }

920 **break** ;

921 **case** RETURN :

922 cnt++;

923 print\_ast (from ,"statement" ,cnt ,"return" ,1);

924 fprint\_token (output);

925 print\_blank ();

926 printf ("return语 句 :\n");

927 gonext ();

928 in ();

929 print\_ast (from ,"statement" ,cnt ,"altexpr" ,0);

930 altexpr ();

931 out ();

932 **if** (nexttype==SEMICON) 933 {

934 cnt++;

935 print\_ast (from ,"statement" ,cnt ," ; " ,1);

936 fprint\_token (output);

937 gonext ();

938 }

939 **else** 940 {

941 synerr (nexttype ,1,SEMICON); 942 }

华 中 科 技 大 学 课 程 实 验 报 告

943

944

945

946

**break** ;

**default** :

{

in ();

947 print\_ast (from ,"statement" ,cnt ," altexpr" ,0);

948 altexpr ();

949 out ();

950 **if** (nexttype==SEMICON) 951 {

|  |  |  |
| --- | --- | --- |
| 952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968 | }  **void**  { | cnt++;  print\_ast (from ,"statement  " ,cnt ," ;" ,1);  fprint\_token (output);  gonext ();  }  **else**  {  synerr (nexttype ,1,SEMICON  );  }  }  }  whilestat ()  **int** from=cnt;  **if** (nexttype==LPAREN)  { |

969 cnt++;

970 print\_ast (from ,"whilestat" ,cnt ," (" ,1);

971 fprint\_token (output);

972 print\_blank ();

973 printf ("循 环 条 件 ： \n");

974 gonext ();

华 中 科 技 大 学 课 程 实 验 报 告

975 in ();

976 print\_ast (from ,"whilestat" ,cnt ,"altexpr" ,0);

977 altexpr ();

978 out ();

979 **if** (nexttype==RPAREN) 980 {

981 cnt++;

982 print\_ast (from ,"whilestat" ,cnt ,")" ,1);

983 fprint\_token (output);

984 gonext ();

985 block\_in ();

986 in ();

987 print\_ast (from ,"whilestat" ,cnt ,"block" ,0)

;

988 block ();

989 out ();

990 block\_out (); 991 }

992 **else** 993 {

994 synerr (nexttype ,1,RPAREN); 995 }

996 }

997 **else**

998 {

999 synerr (nexttype ,1,LPAREN);

1000 }

1001 }

1002 **void** forstat () 1003 {

1004 **int** from=cnt;

1005 print\_blank ();

1006 printf (" 复 合 语 句 ： \n");

1007 **if** (nexttype==LPAREN) 1008 {

华 中 科 技 大 学 课 程 实 验 报 告

1009 cnt++;

1010 print\_ast (from ,"forstat" ,cnt ," (" ,1);

1011 isinfor=1;

1012 fprint\_token (output);

1013 gonext ();

1014 in ();

1015 print\_ast (from ,"forstat" ,cnt ,"forinit" ,0);

1016 forinit ();

1017 out ();

1018 in ();

1019 print\_ast (from ,"forstat" ,cnt ,"altexpr" ,0);

1020 altexpr ();

1021 out ();

1022 **if** (nexttype==SEMICON) 1023 {

1024 cnt++;

1025 print\_ast (from ,"forstat" ,cnt ," ;" ,1);

1026 fprint\_token (output);

1027 gonext ();

1028 in ();

1029 print\_ast (from ,"forstat" ,cnt ,"altexpr" ,0)

;

1030 altexpr ();

1031 out ();

1032 **if** (nexttype==RPAREN) 1033 {

1034 cnt++;

1035 print\_ast (from ,"forstat" ,cnt ,")"

,1);

1036 isinfor=0;

1037 fprint\_token (output);

1038 gonext ();

1039 block\_in ();

1040 in ();

1041 print\_ast (from ,"forstat" ,cnt ,"

华 中 科 技 大 学 课 程 实 验 报 告

block" ,0);

1042 block ();

1043 out ();

1044 block\_out (); 1045 }

1046 **else** 1047 {

1048 synerr (nexttype ,1,RPAREN); 1049 }

1050 }

1051 **else** 1052 {

1053 synerr (nexttype ,1,SEMICON); 1054 }

1055 }

1056 **else** 1057 {

1058 synerr (nexttype ,1,LPAREN); 1059 }

1060 }

1061 **void** forinit () 1062 {

1063 **int** from=cnt;

1064 **if** (nexttype==INT | |nexttype==FLOAT | |nexttype==DOUBLE | | nexttype==CHAR | |nexttype==VOID)

1065 {

1066 in ();

1067 print\_ast (from ,"forinit" ,cnt ,"type" ,0);

1068 type ();**//**应 该 有 一 个**type** **()** **;**

1069 out ();

1070 in ();

1071 print\_ast (from ,"forinit" ,cnt ,"def" ,0);

1072 def ();

1073 out (); 1074 }

华 中 科 技 大 学 课 程 实 验 报 告

1075 **else** 1076 {

1077 in ();

1078 print\_ast (from ,"forinit" ,cnt ,"altexpr" ,0);

1079 altexpr ();

1080 out ();

1081 **if** (nexttype==SEMICON) 1082 {

1083 cnt++;

1084 print\_ast (from ,"forinit" ,cnt ," ;" ,1);

1085 fprint\_token (output);

1086 gonext ();

1087 }

1088 **else** 1089 {

1090 synerr (nexttype ,1,SEMICON); 1091 }

1092 }

1093 }

1094 **void** dowhilestat () 1095 {

1096 **int** from=cnt;

1097 block\_in ();

1098 in ();

1099 print\_ast (from ,"dowhilestat" ,cnt ,"block" ,0);

1100 block ();

1101 out ();

1102 block\_out ();

1103 **if** (nexttype==WHILE) 1104 {

1105 cnt++;

1106 print\_ast (from ,"dowhilestat" ,cnt ,"while" ,1);

1107 fprint\_token (output);

1108 gonext ();

1109 print\_blank ();

华 中 科 技 大 学 课 程 实 验 报 告

1110 printf ("循 环 条 件 ： \n");

1111 **if** (nexttype==LPAREN) 1112 {

1113 cnt++;

1114 print\_ast (from ,"dowhilestat" ,cnt ," (" ,1);

1115 fprint\_token (output);

1116 gonext ();

1117 in ();

1118 print\_ast (from ,"dowhilestat" ,cnt ,"altexpr " ,0);

1119 altexpr ();

1120 out ();

1121 **if** (nexttype==RPAREN) 1122 {

1123 cnt++;

1124 print\_ast (from ,"dowhilestat" ,cnt , ")" ,1);

1125 fprint\_token (output);

1126 gonext ();

1127 **if** (nexttype==SEMICON) 1128 {

1129 cnt++;

1130 print\_ast (from ," dowhilestat" ,cnt ," ;"

,1);

1131 fprint\_token (output);

1132 gonext (); 1133 }

1134 **else** 1135 {

1136 synerr (nexttype ,1,SEMICON

);

1137 }

1138 }

1139 **else**

华 中 科 技 大 学 课 程 实 验 报 告

1140 {

1141 synerr (nexttype ,1,RPAREN); 1142 }

1143 }

1144 **else** 1145 {

1146 synerr (nexttype ,1,LPAREN); 1147 }

1148 }

1149 **else** 1150 {

1151 synerr (nexttype ,1,WHILE); 1152 }

1153 }

1154 **void** ifstat () 1155 {

1156 **int** from=cnt;

1157 **if** (nexttype==LPAREN) 1158 {

1159 cnt++;

1160 print\_ast (from ,"ifstat" ,cnt ," (" ,1);

1161 fprint\_token (output);

1162 print\_blank ();

1163 printf ("条 件 ： \n");

1164 gonext ();

1165 in ();

1166 print\_ast (from ,"ifstat" ,cnt ,"expr" ,0);

1167 expr ();

1168 out ();

1169 **if** (nexttype==RPAREN)

1170 {

1171 cnt++;

1172 print\_ast (from ,"ifstat" ,cnt ,")" ,1);

1173 fprint\_token (output);

1174 gonext ();

华 中 科 技 大 学 课 程 实 验 报 告

1175 block\_in ();

1176 **if** (nexttype==LBRACE) 1177 {

1178 in ();

1179 print\_ast (from ,"ifstat" ,cnt ," block" ,0);

1180 block ();

1181 out (); 1182 }

1183 **else//if** 的 第 二 种 形 式 1184 {

1185 in ();

1186 if2nd=1;

1187 print\_ast (from ,"ifstat" ,cnt ," statement" ,0);

1188 statement ();

1189 out ();

1190 }

1191 block\_out ();

1192 in ();

1193 print\_ast (from ,"ifstat" ,cnt ,"elsestat" ,0)

;

1194 elsestat ();

1195 out (); 1196 }

1197 **else** 1198 {

1199 synerr (nexttype ,1,RPAREN); 1200 }

1201 }

1202 **else** 1203 {

1204 synerr (nexttype ,1,LPAREN); 1205 }

1206 }

华 中 科 技 大 学 课 程 实 验 报 告

1207 **void** elsestat () 1208 {

1209 **int** from=cnt;

1210 print\_blank ();

1211 printf ("else语 句 ： \n");

1212 **if** (nexttype==ELSE)

1213 {

1214 cnt++;

1215 print\_ast (from ,"elsestat" ,cnt ,"else" ,1);

1216 fprint\_token (output);

1217 gonext ();

1218 **if** (nexttype==IF)**//else** **if** 1219 {

1220 cnt++;

1221 print\_ast (from ,"elsestat" ,cnt ,"if" ,1);

1222 fprint\_token (output);

1223 gonext ();

1224 in ();

1225 print\_ast (from ,"elsestat" ,cnt ,"ifstat" ,0)

;

1226 ifstat ();

1227 out (); 1228 }

1229 **else** 1230 {

1231 block\_in ();

1232 **if** (nexttype==LBRACE) 1233 {

1234 in ();

1235 print\_ast (from ,"elsestat" ,cnt ," block" ,0);

1236 block ();

1237 out (); 1238 }

1239 **else//if** 的 第 二 种 形 式

华 中 科 技 大 学 课 程 实 验 报 告

1240 {

1241 in ();

1242 if2nd=1;

1243 print\_ast (from ,"elsestat" ,cnt ," statement" ,0);

1244 statement ();

1245 out ();

1246 }

1247 block\_out (); 1248 }

1249

1250 }

1251 }

1252 **void** switchstat () 1253 {

1254 **int** from=cnt;

1255 **if** (nexttype==LPAREN) 1256 {

1257 cnt++;

1258 print\_ast (from ,"switchstat" ,cnt ," (" ,1);

1259 fprint\_token (output);

1260 print\_blank ();

1261 printf ("switch 变 量 ： \n");

1262 gonext ();

1263 in ();

1264 print\_ast (from ,"switchstat" ,cnt ,"expr" ,0);

1265 expr ();

1266 out ();

1267 **if** (nexttype==RPAREN)

1268 {

1269 cnt++;

1270 print\_ast (from ,"switchstat" ,cnt ,")" ,1);

1271 fprint\_token (output);

1272 gonext ();

1273 block\_in ();

华 中 科 技 大 学 课 程 实 验 报 告

1274 **if** (nexttype==LBRACE) 1275 {

1276 cnt++;

1277 print\_ast (from ,"switchstat" ,cnt ," {" ,1);

1278 fprint\_token (output);

1279 gonext ();

1280 in ();

1281 print\_ast (from ,"switchstat" ,cnt ," casestat" ,0);

1282 casestat ();

1283 out ();

1284 **if** (nexttype==RBRACE) 1285 {

1286 cnt++;

1287 print\_ast (from ," switchstat" ,cnt ,"}" ,1)

;

1288 fprint\_token (output);

1289 gonext (); 1290 }

1291 **else** 1292 {

1293 synerr (nexttype ,1,RBRACE)

;

1294 }

1295 block\_out (); 1296 }

1297 **else** 1298 {

1299 synerr (nexttype ,1,LBRACE); 1300 }

1301 }

1302 **else**

1303 {

华 中 科 技 大 学 课 程 实 验 报 告

1304 synerr (nexttype ,1,RPAREN);

1305 }

1306 }

1307 **else** 1308 {

1309 synerr (nexttype ,1,LPAREN); 1310 }

1311 }

1312 **void** casestat () 1313 {

1314 **int** from=cnt;

1315 **if** (nexttype==CASE) 1316 {

1317 cnt++;

1318 print\_ast (from ,"casestat" ,cnt ,"case" ,1);

1319 fprint\_token (output);

1320 print\_blank ();

1321 printf ("case语 句 ： \n");

1322 gonext ();

1323 print\_blank ();

1324 printf ("编 号 ： \n");

1325 in ();

1326 print\_ast (from ,"casestat" ,cnt ,"literal" ,0);

1327 literal ();

1328 out ();

1329 **if** (nexttype==COLON) 1330 {

1331 cnt++;

1332 print\_ast (from ,"casestat" ,cnt ," :" ,1);

1333 fprint\_token (output);

1334 gonext ();

1335 in ();

1336 print\_ast (from ,"casestat" ,cnt ,"subprogram " ,0);

1337 subprogram ();

华 中 科 技 大 学 课 程 实 验 报 告

1338 out ();

1339 in ();

1340 print\_ast (from ,"casestat" ,cnt ,"casestat"

,0);

1341 casestat ();

1342 out (); 1343 }

1344 **else** 1345 {

1346 synerr (nexttype ,1,COLON); 1347 }

1348 }

1349 **else** **if** (nexttype==DEFAULT) 1350 {

1351 cnt++;

1352 print\_ast (from ,"casestat" ,cnt ,"default" ,1);

1353 fprint\_token (output);

1354 print\_blank ();

1355 printf ("default语 句 ： \n");

1356 gonext ();

1357 **if** (nexttype==COLON) 1358 {

1359 cnt++;

1360 print\_ast (from ,"casestat" ,cnt ," :" ,1);

1361 fprint\_token (output);

1362 gonext ();

1363 in ();

1364 print\_ast (from ,"casestat" ,cnt ,"subprogram " ,0);

1365 subprogram ();

1366 out ();

1367 }

1368 **else**

1369 {

1370 synerr (nexttype ,1,COLON);

华 中 科 技 大 学 课 程 实 验 报 告

1371 }

1372 }

1373 **else** **if**(nexttype==RBRACE){**return** ;}**//case** 的 结 束 条 件

1374 **else** 1375 {

1376 synerr (nexttype ,2,DEFAULT ,CASE); 1377 }

1378 }

1379 **//\*\*\*\*\*\*\*\*\*\*** 表 达 式 函 数 **\*\*\*\*\*\*\*\*\*//**

1380 **void** altexpr () 1381 {

1382 **int** from=cnt;

1383 **if** (nexttype>=LPAREN&&nexttype==SEMICON) 1384 {

1385 **return** ;

1386 }

1387 **else** 1388 {

1389 in ();

1390 print\_ast (from ,"altexpr" ,cnt ,"expr" ,0);

1391 expr ();

1392 out ();

1393 }

1394 }

1395 **void** expr ()

1396 {

1397 **int** from=cnt;

1398 print\_blank ();

1399 printf ("表 达 式 ： \n");

1400 in ();

1401 print\_ast (from ,"expr" ,cnt ,"assexpr" ,0);

1402 assexpr ();

1403 out (); 1404 }

1405 **void** assexpr ()

华 中 科 技 大 学 课 程 实 验 报 告

1406 {

1407 **int** from=cnt;

1408 in ();

1409 print\_ast (from ,"assexpr" ,cnt ,"orexpr" ,0);

1410 orexpr ();

1411 out ();

1412 in ();

1413 print\_ast (from ,"assexpr" ,cnt ,"asstail" ,0);

1414 asstail ();

1415 out (); 1416 }

1417 **void** asstail () 1418 {

1419 **int** from=cnt;

1420 **if** (nexttype==ASSIGN) 1421 {

1422 cnt++;

1423 print\_ast (from ,"asstail" ,cnt ,"=" ,1);

1424 fprint\_token (output);

1425 print\_blank ();

1426 printf ("赋 值\n");

1427 gonext ();

1428 in ();

1429 print\_ast (from ,"asstail" ,cnt ,"orexpr" ,0);

1430 orexpr ();

1431 out ();

1432 in ();

1433 print\_ast (from ,"asstail" ,cnt ,"asstail" ,0);

1434 asstail ();

1435 out ();

1436 }

1437 }

1438 **void** orexpr ()

1439 {

1440 **int** from=cnt;

华 中 科 技 大 学 课 程 实 验 报 告

1441 in ();

1442 print\_ast (from ,"orexpr" ,cnt ,"andexpr" ,0);

1443 andexpr ();

1444 out ();

1445 in ();

1446 print\_ast (from ,"orexpr" ,cnt ,"ortail" ,0);

1447 ortail ();

1448 out (); 1449 }

1450 **void** ortail () 1451 {

1452 **int** from=cnt;

1453 **if** (nexttype==OR) 1454 {

1455 cnt++;

1456 print\_ast (from ,"ortail" ,cnt ," ||" ,1);

1457 fprint\_token (output);

1458 print\_blank ();

1459 printf ("逻 辑 或\n");

1460 gonext ();

1461 in ();

1462 print\_ast (from ,"ortail" ,cnt ,"andexpr" ,0);

1463 andexpr ();

1464 out ();

1465 in ();

1466 print\_ast (from ,"ortail" ,cnt ,"ortail" ,0);

1467 ortail ();

1468 out ();

1469 }

1470 }

1471 **void** andexpr () 1472 {

1473 **int** from=cnt;

1474 in ();

1475 print\_ast (from ,"andexpr" ,cnt ,"cmpexpr" ,0);

华 中 科 技 大 学 课 程 实 验 报 告

1476 cmpexpr ();

1477 out ();

1478 in ();

1479 print\_ast (from ,"andexpr" ,cnt ,"andtail" ,0);

1480 andtail ();

1481 out (); 1482 }

1483 **void** andtail () 1484 {

1485 **int** from=cnt;

1486 **if** (nexttype==AND) 1487 {

1488 cnt++;

1489 print\_ast (from ,"andtail" ,cnt ,"&&" ,1);

1490 fprint\_token (output);

1491 print\_blank ();

1492 printf ("逻 辑 与\n");

1493 gonext ();

1494 in ();

1495 print\_ast (from ,"andtail" ,cnt ,"cmpexpr" ,0);

1496 cmpexpr ();

1497 out ();

1498 in ();

1499 print\_ast (from ,"andtail" ,cnt ,"andtail" ,0);

1500 andtail ();

1501 out ();

1502 }

1503 }

1504 **void** cmpexpr () 1505 {

1506 **int** from=cnt;

1507 in ();

1508 print\_ast (from ,"cmpexpr" ,cnt ,"aloexpr" ,0);

1509 aloexpr ();

1510 out ();

华 中 科 技 大 学 课 程 实 验 报 告

1511 in ();

1512 print\_ast (from ,"cmpexpr" ,cnt ,"cmptail" ,0);

1513 cmptail ();

1514 out (); 1515 }

1516 **void** cmptail () 1517 {

1518 **int** from=cnt;

1519 **if** (nexttype<=NEQU&&nexttype>=GT) 1520 {

1521 fprint\_token (output);

1522 print\_blank ();

1523 **if** (nexttype==GT)

1524 {

1525 cnt++;

1526 print\_ast (from ,"cmptail" ,cnt ,">" ,1);

1527 printf ("大 于\n"); 1528 }

1529 **else** **if** (nexttype==GE) 1530 {

1531 cnt++;

1532 print\_ast (from ,"cmptail" ,cnt ,">=" ,1);

1533 printf ("大 于 等 于\n");

1534

1535 }

1536 **else** **if** (nexttype==LT) 1537 {

1538 cnt++;

1539 print\_ast (from ,"cmptail" ,cnt ,"<" ,1);

1540 printf ("小 于\n");

1541 }

1542 **else** **if** (nexttype==LE) 1543 {

1544 cnt++;

1545 print\_ast (from ,"cmptail" ,cnt ,"<=" ,1);

华 中 科 技 大 学 课 程 实 验 报 告

1546 printf ("小 于 等 于\n"); 1547 }

1548 **else** **if** (nexttype==EQU) 1549 {

1550 cnt++;

1551 print\_ast (from ,"cmptail" ,cnt ,"==" ,1);

1552 printf ("等 于\n"); 1553 }

1554 **else** **if** (nexttype==NEQU) 1555 {

1556 cnt++;

1557 print\_ast (from ,"cmptail" ,cnt ," !=" ,1);

1558 printf ("不 等 于\n"); 1559 }

1560 gonext ();

1561 in ();

1562 print\_ast (from ,"cmptail" ,cnt ,"aloexpr" ,0);

1563 aloexpr ();

1564 out ();

1565 in ();

1566 print\_ast (from ,"cmptail" ,cnt ,"cmptail" ,0);

1567 cmptail ();

1568 out ();

1569 }

1570 }

1571 **void** aloexpr () 1572 {

1573 **int** from=cnt;

1574 in ();

1575 print\_ast (from ,"aloexpr" ,cnt ,"item" ,0);

1576 item ();

1577 out ();

1578 in ();

1579 print\_ast (from ,"aloexpr" ,cnt ,"alotail" ,0);

1580 alotail ();

华 中 科 技 大 学 课 程 实 验 报 告

1581 out (); 1582 }

1583 **void** alotail () 1584 {

1585 **int** from=cnt;

1586 **if** (nexttype==SUB | |nexttype==ADD) 1587 {

1588 fprint\_token (output);

1589 print\_blank ();

1590 **if** (nexttype==SUB)

1591 {

1592 cnt++;

1593 print\_ast (from ,"alotail" ,cnt ,"+" ,1);

1594 printf ("加\n"); 1595 }

1596 **else** **if** (nexttype==ADD) 1597 {

1598 cnt++;

1599 print\_ast (from ,"alotail" ,cnt ,"-" ,1);

1600 printf ("减\n"); 1601 }

1602 gonext ();

1603 in ();

1604 print\_ast (from ,"alotail" ,cnt ,"item" ,0);

1605 item ();

1606 out ();

1607 in ();

1608 print\_ast (from ,"alotail" ,cnt ,"alotail" ,0);

1609 alotail ();

1610 out ();

1611 }

1612 }

1613 **void** item ()

1614 {

1615 **int** from=cnt;

华 中 科 技 大 学 课 程 实 验 报 告

1616 in ();

1617 print\_ast (from ,"item" ,cnt ,"factor" ,0);

1618 factor ();

1619 out ();

1620 in ();

1621 print\_ast (from ,"item" ,cnt ,"itemtail" ,0);

1622 itemtail ();

1623 out (); 1624 }

1625 **void** itemtail () 1626 {

1627 **int** from=cnt;

1628 **if** (nexttype==MUL | |nexttype==MOD | |nexttype==DIV) 1629 {

1630 fprint\_token (output);

1631 print\_blank ();

1632 **if** (nexttype==MUL)

1633 {

1634 cnt++;

1635 print\_ast (from ,"itemtail" ,cnt ,"\*" ,1);

1636 printf ("乘\n"); 1637 }

1638 **else** **if** (nexttype==MOD) 1639 {

1640 cnt++;

1641 print\_ast (from ,"itemtail" ,cnt ,"%" ,1);

1642 printf ("取 余\n"); 1643 }

1644 **else** **if** (nexttype==DIV) 1645 {

1646 cnt++;

1647 print\_ast (from ,"itemtail" ,cnt ,"/" ,1);

1648 printf (" 除 以\n"); 1649 }

1650 gonext ();

华 中 科 技 大 学 课 程 实 验 报 告

1651 in ();

1652 print\_ast (from ,"itemtail" ,cnt ,"factor" ,0);

1653 factor ();

1654 out ();

1655 in ();

1656 print\_ast (from ,"itemtail" ,cnt ,"itemtail" ,0);

1657 itemtail ();

1658 out (); 1659 }

1660 }

1661 **void** factor () 1662 {

1663 **int** from=cnt;

1664 **if** (nexttype==NOT | |nexttype==LEA | |nexttype==MUL | |nexttype ==SUB | |nexttype==INC | |nexttype==DEC)

1665 {

1666 fprint\_token (output);

1667 print\_blank ();

1668 **if** (nexttype==NOT) 1669 {

1670 cnt++;

1671 print\_ast (from ,"factor" ,cnt ," !" ,1);

1672 printf ("逻 辑 非\n");

1673 }

1674 **else** **if** (nexttype==LEA) 1675 {

1676 cnt++;

1677 print\_ast (from ,"factor" ,cnt ,"&" ,1);

1678 printf ("取 地 址\n");

1679 }

1680 **else** **if** (nexttype==MUL) 1681 {

1682 cnt++;

1683 print\_ast (from ,"factor" ,cnt ,"\*" ,1);

1684 printf ("调 用 指 针\n");

华 中 科 技 大 学 课 程 实 验 报 告

1685 }

1686 **else** **if** (nexttype==SUB) 1687 {

1688 cnt++;

1689 print\_ast (from ,"factor" ,cnt ,"-" ,1);

1690 printf ("负 号\n"); 1691 }

1692 **else** **if** (nexttype==INC) 1693 {

1694 cnt++;

1695 print\_ast (from ,"factor" ,cnt ,"++" ,1);

1696 printf (" 自 增\n"); 1697 }

1698 **else** **if** (nexttype==DEC) 1699 {

1700 cnt++;

1701 print\_ast (from ,"factor" ,cnt ,"--" ,1);

1702 printf (" 自 减\n"); 1703 }

1704 gonext ();

1705 in ();

1706 print\_ast (from ,"factor" ,cnt ,"factor" ,0);

1707 factor ();

1708 out ();

1709 }

1710 **else** 1711 {

1712 in ();

1713 print\_ast (from ,"factor" ,cnt ,"val" ,0);

1714 val ();

1715 out ();

1716 }

1717 }

1718 **void** val ()

1719 {

华 中 科 技 大 学 课 程 实 验 报 告

1720 **int** from=cnt;

1721 in ();

1722 print\_ast (from ,"val" ,cnt ,"elem" ,0);

1723 elem ();

1724 out ();

1725 **if** (nexttype==INC | |nexttype==DEC)

1726 {

1727 fprint\_token (output);

1728 print\_blank ();

1729 **if** (nexttype==INC) 1730 {

1731 cnt++;

1732 print\_ast (from ,"val" ,cnt ,"++" ,1);

1733 printf (" 自 增\n"); 1734 }

1735 **else** **if** (nexttype==DEC) 1736 {

1737 cnt++;

1738 print\_ast (from ,"val" ,cnt ,"--" ,1);

1739 printf (" 自 减\n");

1740 }

1741 gonext (); 1742 }

1743 **/\*else**

1744 **{**

1745 **synerr** **()** **;**

1746 **}\*/**

1747 }

1748 **void** elem ()

1749 {

1750 **int** from=cnt;

1751 **if** (nexttype==ID) 1752 {

1753 cnt++;

1754 print\_ast (from ,"elem" ,cnt ,tokenarray [noft-1] .

华 中 科 技 大 学 课 程 实 验 报 告

value ,1);

1755 fprint\_token (output);

1756 print\_blank ();

1757 printf ("名 称 ： %s\n" ,tokenarray [noft-1] .value);

1758 gonext ();

1759 in ();

1760 print\_ast (from ,"elem" ,cnt ,"idexpr" ,0);

1761 idexpr ();

1762 out (); 1763 }

1764 **else** **if** (nexttype==LPAREN) 1765 {

1766 cnt++;

1767 print\_ast (from ,"elem" ,cnt ," (" ,1);

1768 fprint\_token (output);

1769 gonext ();

1770 in ();

1771 expr ();

1772 out ();

1773 **if** (nexttype==RPAREN)

1774 {

1775 cnt++;

1776 print\_ast (from ,"elem" ,cnt ,")" ,1);

1777 fprint\_token (output);

1778 gonext (); 1779 }

1780 **else** 1781 {

1782 synerr (nexttype ,1,RPAREN); 1783 }

1784 }

1785 **else** 1786 {

1787 in ();

1788 print\_ast (from ,"elem" ,cnt ,"literal" ,0);

华 中 科 技 大 学 课 程 实 验 报 告

1789 literal ();

1790 out (); 1791 }

1792 }

1793 **void** idexpr () 1794 {

1795 **int** from=cnt;

1796 **if** (nexttype==LBRACK) 1797 {

1798 cnt++;

1799 print\_ast (from ,"idexpr" ,cnt ," [" ,1);

1800 fprint\_token (output);

1801 gonext ();

1802 in ();

1803 expr ();

1804 out ();

1805 **if** (nexttype==RBRACK)

1806 {

1807 cnt++;

1808 print\_ast (from ,"idexpr" ,cnt ,"]" ,1);

1809 fprint\_token (output);

1810 gonext (); 1811 }

1812 **else** 1813 {

1814 synerr (nexttype ,1,RBRACK); 1815 }

1816 }

1817 **else** **if** (nexttype==LPAREN) 1818 {

1819 cnt++;

1820 print\_ast (from ,"idexpr" ,cnt ," (" ,1);

1821 fprint\_token (output);

1822 print\_blank ();

1823 printf (" 函 数 调 用 的 参 数 ： \n");

华 中 科 技 大 学 课 程 实 验 报 告

1824 gonext ();

1825 in ();

1826 print\_ast (from ,"idexpr" ,cnt ,"realarg" ,0);

1827 realarg ();

1828 out ();

1829 **if** (nexttype==RPAREN) 1830 {

1831 cnt++;

1832 print\_ast (from ,"idexpr" ,cnt ,")" ,1);

1833 fprint\_token (output);

1834 gonext ();

1835 }

1836 **else** 1837 {

1838 synerr (nexttype ,1,RPAREN); 1839 }

1840 }

1841 }

1842 **void** realarg () 1843 {

1844 **int** from=cnt;

1845 **//if** **(nexttype>=LPAREN&&nexttype==SEMICON)**

1846 **if** (nexttype==ID | |nexttype==NUM | |nexttype==STR | |nexttype==

CH)

1847 {

1848 in ();

1849 print\_ast (from ,"realarg" ,cnt ,"expr" ,0);

1850 expr ();

1851 out ();

1852 in ();

1853 print\_ast (from ,"realarg" ,cnt ,"arglist" ,0);

1854 arglist ();

1855 out (); 1856 }

1857 }

华 中 科 技 大 学 课 程 实 验 报 告

1858 **void** arglist () 1859 {

1860 **int** from=cnt;

1861 **if** (nexttype==COMMA) 1862 {

1863 cnt++;

1864 print\_ast (from ,"arglist" ,cnt ," ," ,1);

1865 fprint\_token (output);

1866 gonext ();

1867 in ();

1868 print\_ast (from ,"arglist" ,cnt ,"expr" ,0);

1869 expr ();

1870 out ();

1871 in ();

1872 print\_ast (from ,"arglist" ,cnt ,"arglist" ,0);

1873 arglist ();

1874 out ();

1875 }

1876 }

1877 **void** literal () 1878 {

1879 **int** from=cnt;

1880 fprint\_token (output);

1881 **switch** (nexttype) 1882 {

1883 **case** NUM:

1884 cnt++;

1885 print\_ast (from ,"literal" ,cnt ,tokenarray [ noft-1] .value ,1);

1886 print\_blank ();

1887 printf ("数 值 常 量 ： %s\n" ,tokenarray [noft -1] .value);

1888 gonext ();

1889 **break** ;

1890 **case** STR:

华 中 科 技 大 学 课 程 实 验 报 告

1891 cnt++;

1892 print\_ast (from ,"literal" ,cnt ,tokenarray [ noft-1] .value ,1);

1893 print\_blank ();

1894 printf (" 字 符 串 常 量 ： %s\n" ,tokenarray [noft -1] .value);

1895 gonext ();

1896 **break** ;

1897 **case** CHAR:

1898 cnt++;

1899 print\_ast (from ,"literal" ,cnt ,tokenarray [ noft-1] .value ,1);

1900 print\_blank ();

1901 printf (" 字 符 常 量 ： %s\n" ,tokenarray [noft -1] .value);

1902 gonext ();

1903 **break** ; 1904 }

1905 }

1906 **//\*\*\*\*\*\*\*\*\*\*\*\*\***错 误 处 理 函 数 **\*\*\*\*\*\*\*\*\*\*\*//**

1907 **void** synerr (**int** type ,**int** num , . . .) 1908 {

1909 **int** i=0,expect=0,sign=WRONG ;**//**期 望 出 现 的 词 法 符 号。

1910 va\_list p;

1911 va\_start (p,num);

1912 iserr=1;

1913 **switch** (type) 1914 {

1915 **case** ID:

1916 **for**(i=0;i<num;i++) 1917 {

1918 expect=va\_arg (p,**int**);

1919 **if**(expect<=VOID&&expect>=INT)**//**所

有 可 以 出 现 在 **ID**前 面 的 词 法 符 号。

1920 {

华 中 科 技 大 学 课 程 实 验 报 告

1921 printerr (TYPELOST);

1922 sign=LOST;

1923 **break** ; 1924 }

1925 **else** **if** (expect==ASSIGN) 1926 {

1927 printerr (ASSIGNLOST);

1928 sign=LOST;

1929 **break** ; 1930 }

1931 **else** **if** (expect==LPAREN) 1932 {

1933 printerr (LPARENLOST);

1934 sign=LOST;

1935 **break** ; 1936 }

1937 **else** **if** (expect==RPAREN) 1938 {

1939 printerr (RPARENLOST);

1940 sign=LOST;

1941 **break** ; 1942 }

1943 **else** **if** (expect==LBRACK) 1944 {

1945 printerr (LBRACKLOST);

1946 sign=LOST;

1947 **break** ; 1948 }

1949 **else** **if** (expect==RBRACK) 1950 {

1951 printerr (RBRACKLOST);

1952 sign=LOST;

1953 **break** ; 1954 }

1955 **else** **if** (expect==LBRACE)

华 中 科 技 大 学 课 程 实 验 报 告

1956 {

1957 printerr (LBRACELOST);

1958 sign=LOST;

1959 **break** ; 1960 }

1961 **else** **if** (expect==RBRACE) 1962 {

1963 printerr (RBRACELOST);

1964 sign=LOST;

1965 **break** ; 1966 }

1967 **else** **if** (expect==COMMA) 1968 {

1969 printerr (COMMALOST);

1970 sign=LOST;

1971 **break** ; 1972 }

1973 **else** **if** (expect==COLON) 1974 {

1975 printerr (COLONLOST);

1976 sign=LOST;

1977 **break** ; 1978 }

1979 **else** **if** (expect==SEMICON) 1980 {

1981 printerr (SEMICONLOST);

1982 sign=LOST;

1983 **break** ; 1984 }

1985 }

1986 **if** (sign==WRONG) 1987 {

1988 printerr (IDWRONG); 1989 }

1990 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

1991 **case** INT:

1992 **case** FLOAT :

1993 **case** DOUBLE :

1994 **case** CHAR:

1995 **case** VOID:

1996 **case** EXTERN :

1997 **for**(i=0;i<num;i++) 1998 {

1999 expect=va\_arg (p,**int**);

2000 **if** (expect==LPAREN) 2001 {

2002 printerr (LPARENLOST);

2003 sign=LOST;

2004 **break** ; 2005 }

2006 **else** **if** (expect==RPAREN) 2007 {

2008 printerr (RPARENLOST);

2009 sign=LOST;

2010 **break** ; 2011 }

2012 **else** **if** (expect==LBRACK) 2013 {

2014 printerr (LBRACKLOST);

2015 sign=LOST;

2016 **break** ; 2017 }

2018 **else** **if** (expect==RBRACK) 2019 {

2020 printerr (RBRACKLOST);

2021 sign=LOST;

2022 **break** ; 2023 }

2024 **else** **if** (expect==LBRACE) 2025 {

华 中 科 技 大 学 课 程 实 验 报 告

2026 printerr (LBRACELOST);

2027 sign=LOST;

2028 **break** ; 2029 }

2030 **else** **if** (expect==RBRACE) 2031 {

2032 printerr (RBRACELOST);

2033 sign=LOST;

2034 **break** ; 2035 }

2036 **else** **if** (expect==COLON) 2037 {

2038 printerr (COLONLOST);

2039 sign=LOST;

2040 **break** ; 2041 }

2042 **else** **if** (expect==SEMICON) 2043 {

2044 printerr (SEMICONLOST);

2045 sign=LOST;

2046 **break** ; 2047 }

2048 }

2049 **if** (sign==WRONG)

2050 {

2051 printerr (TYPEWRONG); 2052 }

2053 **break** ;

2054 **case** NUM:

2055 **case** CH:

2056 **case** STR:

2057 **for**(i=0;i<num;i++) 2058 {

2059 expect=va\_arg (p,**int**);

2060 **if** (expect==ASSIGN)

华 中 科 技 大 学 课 程 实 验 报 告

2061 {

2062 printerr (ASSIGNLOST);

2063 sign=LOST;

2064 **break** ; 2065 }

2066 **else** **if** (expect==LPAREN) 2067 {

2068 printerr (LPARENLOST);

2069 sign=LOST;

2070 **break** ; 2071 }

2072 **else** **if** (expect==LBRACK) 2073 {

2074 printerr (LBRACKLOST);

2075 sign=LOST;

2076 **break** ; 2077 }

2078 }

2079 **if** (sign==WRONG) 2080 {

2081 printerr (LITERALWRONG); 2082 }

2083 **break** ;

2084 **case** ADD:

2085 **case** SUB:

2086 **case** DIV:

2087 **case** MOD:

2088 **case** GT:

2089 **case** GE:

2090 **case** LT:

2091 **case** LE:

2092 **case** EQU:

2093 **case** NEQU:

2094 **case** AND:

2095 **case** OR:

华 中 科 技 大 学 课 程 实 验 报 告

2096 **for**(i=0;i<num;i++) 2097 {

2098 expect=va\_arg (p,**int**);

2099 **if** (expect==NUM | |expect==CH | |

expect==STR)

2100 {

2101 printerr (LITERALLOST);

2102 sign=LOST;

2103 **break** ; 2104 }

2105 **else** **if** (expect==ID) 2106 {

2107 printerr (IDLOST);

2108 sign=LOST;

2109 **break** ; 2110 }

2111 }

2112 **break** ;

2113 **case** NOT:

2114 **case** LEA:

2115 **for**(i=0;i<num;i++) 2116 {

2117 expect=va\_arg (p,**int**);

2118 **if** (expect==ASSIGN) 2119 {

2120 printerr (ASSIGNLOST);

2121 sign=LOST;

2122 **break** ; 2123 }

2124 **else** **if** (expect==LPAREN) 2125 {

2126 printerr (LPARENLOST);

2127 sign=LOST;

2128 **break** ; 2129 }

华 中 科 技 大 学 课 程 实 验 报 告

2130 **else** **if** (expect==COMMA) 2131 {

2132 printerr (COMMALOST);

2133 sign=LOST;

2134 **break** ; 2135 }

2136 **else** **if** (expect==COLON) 2137 {

2138 printerr (COLONLOST);

2139 sign=LOST;

2140 **break** ; 2141 }

2142 **else** **if** (expect==SEMICON) 2143 {

2144 printerr (SEMICONLOST);

2145 sign=LOST;

2146 **break** ; 2147 }

2148 }

2149 **break** ;

2150 **case** MUL:

2151 **for**(i=0;i<num;i++) 2152 {

2153 expect=va\_arg (p,**int**);

2154 **if** (expect==NUM | |expect==CH | |

expect==STR)

2155 {

2156 printerr (LITERALLOST);

2157 sign=LOST;

2158 **break** ; 2159 }

2160 **else** **if** (expect==ID) 2161 {

2162 printerr (IDLOST);

2163 sign=LOST;

华 中 科 技 大 学 课 程 实 验 报 告

2164 **break** ; 2165 }

2166 **if** (expect<=VOID&&expect>=INT) 2167 {

2168 printerr (TYPELOST);

2169 sign=LOST;

2170 **break** ; 2171 }

2172 **else** **if** (expect==ASSIGN) 2173 {

2174 printerr (ASSIGNLOST);

2175 sign=LOST;

2176 **break** ; 2177 }

2178 **else** **if** (expect==LPAREN) 2179 {

2180 printerr (LPARENLOST);

2181 sign=LOST;

2182 **break** ; 2183 }

2184 **else** **if** (expect==RPAREN) 2185 {

2186 printerr (RPARENLOST);

2187 sign=LOST;

2188 **break** ; 2189 }

2190 **else** **if** (expect==LBRACK) 2191 {

2192 printerr (LBRACKLOST);

2193 sign=LOST;

2194 **break** ; 2195 }

2196 **else** **if** (expect==RBRACK) 2197 {

2198 printerr (RBRACKLOST);

sign=LOST;

华 中 科 技 大 学 课 程 实 验 报 告

2199

2200

2201

2202

2203

2204

2205

2206

2207

2208

2209

2210

2211

2212

2213

2214

2215

2216

2217

2218

2219

2220

2221

2222

2223

2224

2225

2226

2227

2228

2229

2230

2231

2232

2233

}

**else**

{

}

**else**

{

}

**else**

{

}

**else**

{

}

**else**

{

}

}

**break** ;

**break** ;

**if** (expect==LBRACE)

printerr (LBRACELOST);

sign=LOST;

**break** ;

**if** (expect==RBRACE)

printerr (RBRACELOST);

sign=LOST;

**break** ;

**if** (expect==COMMA)

printerr (COMMALOST);

sign=LOST;

**break** ;

**if** (expect==COLON)

printerr (COLONLOST);

sign=LOST;

**break** ;

**if** (expect==SEMICON)

printerr (SEMICONLOST);

sign=LOST;

**break** ;

华 中 科 技 大 学 课 程 实 验 报 告

2234 **case** INC:

2235 **case** DEC:

2236 **for**(i=0;i<num;i++) 2237 {

2238 expect=va\_arg (p,**int**);

2239 **if** (expect==ID)

2240 {

2241 printerr (IDLOST);

2242 sign=LOST;

2243 **break** ; 2244 }

2245 **else** **if** (expect==ASSIGN) 2246 {

2247 printerr (ASSIGNLOST);

2248 sign=LOST;

2249 **break** ; 2250 }

2251 **else** **if** (expect==LPAREN) 2252 {

2253 printerr (LPARENLOST);

2254 sign=LOST;

2255 **break** ; 2256 }

2257 **else** **if** (expect==RPAREN) 2258 {

2259 printerr (RPARENLOST);

2260 sign=LOST;

2261 **break** ; 2262 }

2263 **else** **if** (expect==LBRACK) 2264 {

2265 printerr (LBRACKLOST);

2266 sign=LOST;

2267 **break** ; 2268 }

华 中 科 技 大 学 课 程 实 验 报 告

2269 **else** **if** (expect==RBRACK) 2270 {

2271 printerr (RBRACKLOST);

2272 sign=LOST;

2273 **break** ; 2274 }

2275 **else** **if** (expect==LBRACE) 2276 {

2277 printerr (LBRACELOST);

2278 sign=LOST;

2279 **break** ; 2280 }

2281 **else** **if** (expect==RBRACE) 2282 {

2283 printerr (RBRACELOST);

2284 sign=LOST;

2285 **break** ; 2286 }

2287 **else** **if** (expect==COMMA) 2288 {

2289 printerr (COMMALOST);

2290 sign=LOST;

2291 **break** ; 2292 }

2293 **else** **if** (expect==COLON) 2294 {

2295 printerr (COLONLOST);

2296 sign=LOST;

2297 **break** ; 2298 }

2299 **else** **if** (expect==SEMICON) 2300 {

2301 printerr (SEMICONLOST);

2302 sign=LOST;

2303 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

2304 }

2305 }

2306 **break** ;

2307 **case** LPAREN :

2308 **for**(i=0;i<num;i++) 2309 {

2310 expect=va\_arg (p,**int**);

2311 **if** (expect==ID)

2312 {

2313 printerr (IDLOST);

2314 sign=LOST;

2315 **break** ; 2316 }

2317 **else** **if** (expect==ASSIGN) 2318 {

2319 printerr (ASSIGNLOST);

2320 sign=LOST;

2321 **break** ; 2322 }

2323 **else** **if** (expect<=RETURN&&expect>=

IF)

2324 {

2325 printerr (STATEMENTLOST);

2326 sign=LOST;

2327 **break** ; 2328 }

2329 **else** **if** (expect==LPAREN) 2330 {

2331 printerr (LPARENLOST);

2332 sign=LOST;

2333 **break** ; 2334 }

2335 **else** **if** (expect==LBRACK) 2336 {

2337 printerr (LBRACKLOST);

华 中 科 技 大 学 课 程 实 验 报 告

2338 sign=LOST;

2339 **break** ; 2340 }

2341 }

2342 **if** (sign==WRONG) 2343 {

2344 printerr (LPARENWRONG); 2345 }

2346 **break** ;

2347 **case** RPAREN :

2348 **for**(i=0;i<num;i++) 2349 {

2350 expect=va\_arg (p,**int**);

2351 **if** (expect==ID)

2352 {

2353 printerr (IDLOST);

2354 sign=LOST;

2355 **break** ; 2356 }

2357 **else** **if** (expect==NUM | |expect==CH | |

expect==STR)

2358 {

2359 printerr (LITERALLOST);

2360 sign=LOST;

2361 **break** ; 2362 }

2363 **else** **if** (expect==LPAREN) 2364 {

2365 printerr (LPARENLOST);

2366 sign=LOST;

2367 **break** ; 2368 }

2369 **else** **if** (expect==RPAREN) 2370 {

2371 printerr (RPARENLOST);

华 中 科 技 大 学 课 程 实 验 报 告

2372 sign=LOST;

2373 **break** ; 2374 }

2375 }

2376 **if** (sign==WRONG) 2377 {

2378 printerr (RPARENWRONG); 2379 }

2380 **break** ;

2381 **case** LBRACK :

2382 **for**(i=0;i<num;i++) 2383 {

2384 expect=va\_arg (p,**int**);

2385 **if** (expect==ID)

2386 {

2387 printerr (IDLOST);

2388 sign=LOST;

2389 **break** ; 2390 }

2391 }

2392 **if** (sign==WRONG) 2393 {

2394 printerr (LBRACKWRONG); 2395 }

2396 **break** ;

2397 **case** RBRACK :

2398 **for**(i=0;i<num;i++) 2399 {

2400 expect=va\_arg (p,**int**);

2401 **if** (expect==NUM)

2402 {

2403 printerr (NUMLOST);

2404 sign=LOST;

2405 **break** ; 2406 }

华 中 科 技 大 学 课 程 实 验 报 告

2407 **else** **if** (expect==RPAREN) 2408 {

2409 printerr (RPARENLOST);

2410 sign=LOST;

2411 **break** ; 2412 }

2413 }

2414 **if** (sign==WRONG) 2415 {

2416 printerr (LBRACKWRONG); 2417 }

2418 **break** ;

2419 **case** LBRACE :

2420 **for**(i=0;i<num;i++) 2421 {

2422 expect=va\_arg (p,**int**);

2423 **if** (expect==ID)

2424 {

2425 printerr (IDLOST);

2426 sign=LOST;

2427 **break** ; 2428 }

2429 **else** **if** (expect==SEMICON) 2430 {

2431 printerr (SEMICONLOST);

2432 sign=LOST;

2433 **break** ; 2434 }

2435 **else** **if** (expect==RPAREN) 2436 {

2437 printerr (RPARENLOST);

2438 sign=LOST;

2439 **break** ; 2440 }

2441 **else** **if** (expect==LBRACE)

华 中 科 技 大 学 课 程 实 验 报 告

2442 {

2443 printerr (LBRACELOST);

2444 sign=LOST;

2445 **break** ; 2446 }

2447 **else** **if** (expect==RBRACE) 2448 {

2449 printerr (RBRACELOST);

2450 sign=LOST;

2451 **break** ; 2452 }

2453 }

2454 **if** (sign==WRONG) 2455 {

2456 printerr (LBRACKWRONG); 2457 }

2458 **break** ;

2459 **case** RBRACE :

2460 **for**(i=0;i<num;i++) 2461 {

2462 expect=va\_arg (p,**int**);

2463 **if** (expect==SEMICON) 2464 {

2465 printerr (SEMICONLOST);

2466 sign=LOST;

2467 **break** ; 2468 }

2469 **else** **if** (expect==RPAREN) 2470 {

2471 printerr (RPARENLOST);

2472 sign=LOST;

2473 **break** ; 2474 }

2475 **else** **if** (expect==RBRACK) 2476 {

华 中 科 技 大 学 课 程 实 验 报 告

2477 printerr (RBRACKLOST);

2478 sign=LOST;

2479 **break** ; 2480 }

2481 **else** **if** (expect==LBRACE) 2482 {

2483 printerr (LBRACELOST);

2484 sign=LOST;

2485 **break** ; 2486 }

2487 **else** **if** (expect==RBRACE) 2488 {

2489 printerr (RBRACELOST);

2490 sign=LOST;

2491 **break** ; 2492 }

2493 }

2494 **if** (sign==WRONG) 2495 {

2496 printerr (RBRACEWRONG); 2497 }

2498 **break** ;

2499 **case** IF:

2500 **case** ELSE:

2501 **case** SWITCH :

2502 **case** CASE:

2503 **case** DEFAULT :

2504 **case** WHILE :

2505 **case** DO:

2506 **case** FOR:

2507 **case** BREAK :

2508 **case** CONTINUE :

2509 **case** RETURN :

2510 **for**(i=0;i<num;i++) 2511 {

华 中 科 技 大 学 课 程 实 验 报 告

2512

2513

2514

2515

2516

2517

2518

2519

2520

2521

2522

2523

2524

2525

2526

2527

2528

2529

2530

2531

2532

2533

2534

2535

2536

2537

2538

2539

2540

2541

2542

2543

2544

2545

2546

expect=va\_arg (p,**int**);

**if** (expect==SEMICON)

{

printerr (SEMICONLOST);

sign=LOST;

**break** ;

}

**else** **if** (expect==LBRACE) {

printerr (LBRACELOST);

sign=LOST;

**break** ;

}

**else** **if** (expect==LBRACE) {

printerr (RBRACELOST);

sign=LOST;

**break** ;

}

**else** **if** (expect==RBRACE) {

printerr (RBRACELOST);

sign=LOST;

**break** ;

}

}

**if** (sign==WRONG)

{

printerr (STATEMENTWRONG); }

**break** ;

**default** :

printf (" 第%d 行 ： 未 知 错 误！ \n" ,loc);

**//exit** **(-1)** **;**

}

华 中 科 技 大 学 课 程 实 验 报 告

2547 va\_end (p);

2548 **if**(sign==WRONG)**//**如 果 是 符 号 错 误 ， 继 续 读 取 下 一 个**token** 2549 {

2550 gonext (); 2551 }

2552 exit (-1);

2553 }

2554 **void** printerr (**int** errtype) **//**用 于 显 示 语 法 错 误 的 名 称 2555 {

2556 **switch** (errtype) 2557 {

2558 **case** TYPELOST :

2559 printf (" 第%d 行 ： 类 型 缺 失！ \n" ,loc);

2560 **break** ;

2561 **case** TYPEWRONG :

2562 printf (" 第%d 行 ： 类 型 匹 配 失 败！ \n" ,loc);

2563 **break** ;

2564 **case** IDLOST :

2565 printf (" 第%d 行 ： 标 识 符 缺 失！ \n" ,loc);

2566 **break** ;

2567 **case** IDWRONG :

2568 printf (" 第%d 行 ： 标 识 符 匹 配 失 败！ \n" ,loc);

2569 **break** ;

2570 **case** NUMLOST :

2571 printf (" 第%d 行 ： 数 组 的 长 度 缺 失！ \n" ,loc);

2572 **break** ;

2573 **case** NUMWRONG :

2574 printf (" 第%d 行 ： 数 组 的 长 度 匹 配 失 败！ \n" , loc);

2575 **break** ;

2576 **case** LITERALLOST :

2577 printf (" 第%d 行 ： 常 量 缺 失！ \n" ,loc);

2578 **break** ;

2579 **case** LITERALWRONG :

2580 printf (" 第%d 行 ： 常 量 匹 配 失 败！ \n" ,loc);

华 中 科 技 大 学 课 程 实 验 报 告

2581 **break** ;

2582 **case** COMMALOST :

2583 printf (" 第%d 行 ： 逗 号 缺 失！ \n" ,loc);

2584 **break** ;

2585 **case** COMMAWRONG :

2586 printf (" 第%d 行 ： 逗 号 匹 配 失 败！ \n" ,loc);

2587 **break** ;

2588 **case** SEMICONLOST :

2589 printf (" 第%d 行 ： 分 号 缺 失！ \n" ,loc);

2590 **break** ;

2591 **case** SEMICONWRONG :

2592 printf (" 第%d 行 ： 分 号 匹 配 失 败！ \n" ,loc);

2593 **break** ;

2594 **case** ASSIGNLOST :

2595 printf (" 第%d 行 ： 赋 值 运 算 符 缺 失！ \n" ,loc);

2596 **break** ;

2597 **case** ASSIGNWRONG :

2598 printf (" 第%d 行 ： 赋 值 运 算 符 匹 配 失 败！ \n" , loc);

2599 **break** ;

2600 **case** COLONLOST :

2601 printf (" 第%d 行 ： 冒 号 缺 失！ \n" ,loc);

2602 **break** ;

2603 **case** COLONWRONG :

2604 printf (" 第%d 行 ： 冒 号 匹 配 失 败！ \n" ,loc);

2605 **break** ;

2606 **case** STATEMENTLOST :

2607 printf (" 第%d 行 ： 语 句 缺 失！ \n" ,loc);

2608 **break** ;

2609 **case** STATEMENTWRONG :

2610 printf (" 第%d 行 ： 语 句 匹 配 失 败！ \n" ,loc);

2611 **break** ;

2612 **case** LPARENLOST :

2613 printf (" 第%d 行 ： 左 小 括 号 缺 失！ \n" ,loc);

2614 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

2615 **case** LPARENWRONG :

2616 printf (" 第%d 行 ： 左 小 括 号 匹 配 失 败！ \n" ,loc );

2617 **break** ;

2618 **case** RPARENLOST :

2619 printf (" 第%d 行 ： 右 小 括 号 缺 失！ \n" ,loc);

2620 **break** ;

2621 **case** RPARENWRONG :

2622 printf (" 第%d 行 ： 右 小 括 号 匹 配 失 败！ \n" ,loc );

2623 **break** ;

2624 **case** LBRACKLOST :

2625 printf (" 第%d 行 ： 左 中 括 号 缺 失！ \n" ,loc);

2626 **break** ;

2627 **case** LBRACKWRONG :

2628 printf (" 第%d 行 ： 左 中 括 号 匹 配 失 败！ \n" ,loc );

2629 **break** ;

2630 **case** RBRACKLOST :

2631 printf (" 第%d 行 ： 右 中 括 号 缺 失！ \n" ,loc);

2632 **break** ;

2633 **case** PBRACKWRONG :

2634 printf (" 第%d 行 ： 右 中 括 号 匹 配 失 败！ \n" ,loc );

2635 **break** ;

2636 **case** LBRACELOST :

2637 printf (" 第%d 行 ： 左 大 括 号 缺 失！ \n" ,loc);

2638 **break** ;

2639 **case** LBRACEWRONG :

2640 printf (" 第%d 行 ： 左 大 括 号 匹 配 失 败！ \n" ,loc );

2641 **break** ;

2642 **case** RBRACELOST :

2643 printf (" 第%d 行 ： 右 大 括 号 缺 失！ \n" ,loc);

2644 **break** ;

华 中 科 技 大 学 课 程 实 验 报 告

2645 **case** RBRACEWRONG :

2646 printf (" 第%d 行 ： 右 大 括 号 匹 配 失 败！ \n" ,loc );

2647 **break** ; 2648 }

2649 }

2650 **#endif**

parser\_head

1 #ifndef \_\_PARSER\_HEAD\_\_

2 **#define** \_\_PARSER\_HEAD\_\_

3 **//\*\*\*\*\*\*\*\*\*\*\***控 制 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

4 **void** readtokenarray (FILE\*fp);**//**读 入**token**

5 **void** gonext ();**//**下 一 个**token**

6 **void** in ();**//**进 入**AST** 中 的 子 树

7 **void** out ();**//**退 出**AST** 中 的 子 树

8 **void** block\_in ();**//**进 入**block**

9 **void** block\_out ();**//**退 出**block**

10 **void** print\_blank ();**//** 屏 幕 打 印 空 格

11 **void** fprint\_token (FILE\*fp);**//**打 印**token** 12 **//\*\*\*\*\*\*\*\*\*\*\***绘 制**AST** 的 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

13 **void** init\_ast ();**//**初 始 化**AST**绘 制文 件 输 出 模 块

14 **void** end\_ast ();**//**结 束**AST**绘 制文 件 输 出 模 块

15 **void** print\_ast (**int** from ,**const** **char**\*from\_name ,**int** to ,**const** **char**\* to\_name ,**int** is\_terminal);**//AST**绘 制 文 件 输 出

16 **//\*\*\*\*\*\*\*\*\*\*** 变 量 声 明 函 数 **\*\*\*\*\*\*\*\*\*\*\*//**

17 **void** program ();**//paser**入 口 函 数

18 **void** segment ();**//**程 序 段

19 **void** type ();**//**类 型

20 **void** def ();**//** 除 去 类 型 的 完 整 声 明

21 **void** init ();**//**初 始 化 赋 值

22 **void** deflist ();**//** ， 后 的 连 续 声 明

23 **void** idtail ();**//** 函 数 声 明

24 **void** defdata ();**//**后 续 声 明 信 息

25 **void** varrdef ();**//**数 组 声 明

26 **//\*\*\*\*\*\*\*\*\*** 函 数 声 明 函 数 **\*\*\*\*\*\*\*\*\*//**

27 **void** funtail ();**//** 函 数 名称 后 的 声 明

28 **void** paralist ();**//**形 参 列 表

29 **void** paradata ();**//** 函 数 形 参

30 **void** para ();**//** 函 数 形 参 声 明

31 **void** paradatatail ();**//**后 续 函 数 形 参

32 **//\*\*\*\*\*\*\*\*\*\*\*\*\*** 函 数 定 义 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*//**

33 **void** block ();**//**块

34 **void** subprogram ();**//{}** 中 的 程 序

35 **void** localdef ();**//**本 地 变 量 声 明

36 **//\*\*\*\*\*\*\*\*\*\*\*\***语 句 函 数 **\*\*\*\*\*\*\*\*\*\*\*\*\*\*//**

37 **void** statement ();**//**语 句 入 口 函 数

38 **void** whilestat ();

39 **void** forstat ();

40 **void** forinit ();

41 **void** dowhilestat ();

42 **void** ifstat ();

43 **void** elsestat ();

44 **void** casestat ();

45 **void** switchstat ();

46 **//\*\*\*\*\*\*\*\*\*\*** 表 达 式 函 数 **\*\*\*\*\*\*\*\*\*//**

47 **void** altexpr ();**//**可 选 的 表 达 式入 口 函 数

48 **void** expr ();**//**表 达 式 入 口 函 数

49 **void** assexpr ();**//**赋 值 运 算

50 **void** asstail ();

51 **void** orexpr ();**//**逻 辑 或 运 算

52 **void** ortail ();

53 **void** andexpr ();**//**逻 辑 与 运 算

54 **void** andtail ();

55 **void** cmpexpr ();**//** 关 系 运 算

56 **void** cmptail ();

57 **void** aloexpr ();**//+-**运 算

58 **void** alotail ();

59 **void** item ();**//** 带 单 目 运 算 的 整 体

60 **void** itemtail ();**//\*/%**运 算

61 **void** factor ();**//**单 目 运 算

62 **void** val ();**//**前 置 自 增 自 减

63 **void** elem ();**//**参 与 运 算 的 最 小 单 元

64 **void** idexpr ();**//**参 与 运 算 的 数 组 或 函 数

65 **void** realarg ();**//**参 与 运 算 的 函 数 形 参

66 **void** arglist ();**//**形 参 列 表

67 **void** literal ();**//** 常 数

68 **//\*\*\*\*\*\*\*\*\*\*\*\*\***错 误 处 理 函 数 **\*\*\*\*\*\*\*\*\*\*//**

69 **void** synerr (**int** type ,**int** num , . . .) ; **//type** **:**实 际 得 到 的 词 法 符 号 。 **num** **:**期 望 得到 的 词 法 符号 的 个 数 。 后 续 是 期望 得到 的 词 法 符 号。

70 **void** printerr (**int** errtype);**//**打 印 错 误。 71

72 **int** noft=0,nofc=0,nexttype=0,loc ,if2nd=0,isinfor=0,iserr=0, blocknum=0,layer=0;**//**初 始 化 用 于 记 录 层 的 栈。

73 **int** cnt=0;**//**用 于 记 录**AST** 节 点 编 号。

74 **int** setted [MAXLEN]={0};**//**用 于 记 录 节 点 是 否 已 设 置。

75 **char** from\_name [MAXLEN\_OF\_CONTENT];

76 **char** to\_name [MAXLEN\_OF\_CONTENT];

77 Token tokenarray [MAXLEN];**//token**序 列。

78 FILE\*tokenlist ;

79 FILE\*output ;

80 FILE\*graph ; 81 **#endif**

parser\_main

1 **#include**"parser\_def .h"

2 **int** main () 3 {

4 tokenlist=fopen (" . ./lexer/temp\_token .txt" ,"r");

5 output=fopen (" . ./result .txt" ,"w");

6 init\_ast ();

7 **if** (!tokenlist) 8 {

9 printf ("未 找 到Token序 列 文 件！ \n");

10 exit (-1);

11 }

12 readtokenarray (tokenlist);

13 fclose (tokenlist);

14 gonext ();

15 program ();

16 end\_ast ();

17 **return** 0; 18 }

**9.4** **AST** 生成工具

openweb.exe

1 #include "stdio .h"

2 #include "stdlib .h"

3 #include<windows .h>

4 #include<string .h>

5 **void** open\_web (**char** \*website) 6 {

7 ShellExecuteA (0,"open" , website ,0 ,0 ,1); 8 }

9

10 **int** main(**int** argc ,**char** \*argv []) 11 {

12 HWND hwnd;

13 hwnd=FindWindow ("ConsoleWindowClass" ,NULL);

14 ShowWindow (hwnd ,SW\_HIDE);

15 open\_web ("http ://dreampuf .github .io/ GraphvizOnline");

16 **return** 0; 17 }

AST\_creator.bat

1 dot -Tpng AST .dot -o AST .png

**10** 附录 **B** 程序使用说明

编译顺序：

1. 选择 1 编译 lexer\_main.c

2. 选择 2 编译 parser\_main.c

运行顺序：

1. 打开总系统.exe

2. 将待处理的代码放入 input.txt

3. 选择 1 进行词法分析

4. 选择 2 进行语法分析

5. 选择 3 进行缩进编排

6. 运行 create\_AST.bat 或者打开 openweb.exe 在线编译

7. 打开 AST.png