

**课程设计报告**

**题目：基于高级语言源程序格式处理工具**

**课程名称：程序设计综合课程设计**

**专业班级： CS1804**

**学 号： U201814589**

**姓 名： 王远航**

**指导教师： 袁凌**

**报告日期： 2020年3月30日**

**计算机科学与技术学院**

**目录**

[1.引言 2](#_Toc4955)

[1.1问题描述 2](#_Toc25180)

[1.2需求与技术现状分析 2](#_Toc1009)

[2.程序总体设计 3](#_Toc22683)

[2.1设计目的 3](#_Toc16081)

[2.2设计要求 3](#_Toc28226)

[2.3总体程序设计思路 4](#_Toc19876)

[2.4流程图 4](#_Toc27283)

[3.数据结构和算法详细设计 5](#_Toc5618)

[3.1词法分析设计 5](#_Toc13052)

[3.2语法分析设计 7](#_Toc19803)

[3.3保存文件设计 7](#_Toc31995)

[4. 系统实现和测试 8](#_Toc14414)

[4.1词法分析 8](#_Toc5228)

[4.1语法分析 9](#_Toc20418)

[4.3保存文件设计 13](#_Toc228)

[4.4.系统测试 14](#_Toc3459)

[5. 总结与展望 23](#_Toc27462)

[5.1全文总结 23](#_Toc23563)

[5.2工作展望 23](#_Toc27841)

[6.体会 24](#_Toc19970)

[6.1总结 24](#_Toc31429)

[6.2特色 24](#_Toc8920)

[6.3不足 25](#_Toc32703)

[参考文献 25](#_Toc2326)

[附录 26](#_Toc28316)

[附录1： 26](#_Toc21595)

[附录2： 70](#_Toc20313)

## 1.引言

### 1.1问题描述

抽象语法树(abstract syntax code, AST) 是源代码的抽象语法结构的树状表示，树上的每个节点都表示源代码中的一种结构， 这所以说是抽象的，是因为抽象语法树并不会表示出真实语法出现的每一个细节， 比如说，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现。抽象语法树并不依赖于源语言的语法，也就是说语法分析阶段所采用的上下文无文文法，因为在写文法时，经常会对文法进行等价的转换(消除左递归，回溯，二义性等) , 这样会给文法分析引入一些多余的成分，对后续阶段造成不利影响，甚至会使合个阶段变得混乱。因些，很多编译器经常要独立地构造语法分析树为前端，后端建立一个清晰的接口。

### 1.2需求与技术现状分析

抽象语法树(Abstract Syntax Tree ,AST)作为程序的一种中间表示形式,在[程序分析](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E5%88%86%E6%9E%90" \t "_blank)等诸多领域有广泛的应用.利用抽象语法树可以方便地实现多种[源程序](https://baike.baidu.com/item/%E6%BA%90%E7%A8%8B%E5%BA%8F" \t "_blank)处理工具,比如源程序浏览器、智能编辑器、[语言翻译器](https://baike.baidu.com/item/%E8%AF%AD%E8%A8%80%E7%BF%BB%E8%AF%91%E5%99%A8/6119005" \t "_blank)等。通常是作为[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91/_blank)或[解释器](https://baike.baidu.com/item/%E8%A7%A3%E9%87%8A%E5%99%A8" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91/_blank)的组件出现的，它的作用是进行语法检查、并构建由输入的单词组成的数据结构（一般是[语法分析树](https://baike.baidu.com/item/%E8%AF%AD%E6%B3%95%E5%88%86%E6%9E%90%E6%A0%91" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91/_blank)、[抽象语法树](https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91/_blank)等层次化的数据结构）。语法分析器通常使用一个独立的[词法分析器](https://baike.baidu.com/item/%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90%E5%99%A8" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E8%AF%AD%E6%B3%95%E6%A0%91/_blank)从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。实际开发中，语法分析器可以手工编写，也可以使用工具（半）自动生成。

个人通过对源程序的实现，发现各种编译器的中间表示就是语法分析树，AST，感觉本次实验的最终目的就是编译出来一个简单的语言编译器，所以我认为语法分析树目前的现状就是编译器的内部原理，现在实现的程序比较简单，词法识别和语法识别还不够全面，只能够识别比较简单的较为基本的源程序。

## 2.程序总体设计

### 2.1设计目的

编写一个具有词法分析（能够识别出各种单词）和语法分析（生成语法分析树AST）和能够将输入的文本程序以一定的格式输出并且保存在文件中。

### 2.2设计要求

要求具有如下功能：

**总体设计功能：**由源程序到抽象语法树的过程，逻辑上包含2个重要的阶段，一是词法分析，识别出所有按词法规则定义的单词；二是语法分析，根据定义的语法规则，分析单词序列是否满足语法规则，同时生成抽象语法树。

**输入输出功能：**本程序没有读取文件的功能，需要在exe文件中输入源代码，然后以ctrl+z进行结尾才能开始，输出时可以进行保存文件，并且保存的文件中的源代码有一定的规范。

**（1）词法分析过程：**词法分析需要识别出五类单词，标识符、关键字、常量、运算符和定界符，词法分析每识别出一个单词，就可返回单词的编码。该系统可以有效地识别关键字，字符，整形数字，长整型数字，变量id，数组，单目操作符号，和双目操作符号，以及宏定义和行注释与段注释

**（2）语法分析生成AST语法树的过程：**通过识别出来的各种字符，采用的实现方法是编译技术中的递归下降子程序法，生成一个识别各种结构的语法树，例如复合语句，函数参数，函数参数列表，关键字int和id的赋值，if—else语句，while循环语句等

### 2.3总体程序设计思路

本程序使用C语言实现，程序分为三部分：词法分析（scan），语法分析（program）和保存文件(savefile)，分别将3个处理阶段写在三个函数中，分别是scan()，program()和savefile(),三个函数分别完成词法分析语法分析和保存文件的任务。scan()函数主要的工作是检查注释是否合法、词法分析获取token\_text。Program()函数的主要工作是根据scan()词法分析之后的token\_text进行语法分析，生成语法树，最后并输出语法树。Savefile()是根据生成的语法分析树，采用先根遍历的方式将程序代码采用一定的格式保存在指定文件里面。

表2.1代码块及功能简要概述

|  |  |
| --- | --- |
| 模块 | 功能 |
| scan()：词法分析 | 将输入的单词进行区别，并且保存原值 |
| program()：语法分析 | 进行语法分析，并且生成语法树 |
| savefile()：保存文件 | 以一定的格式保存文件 |

### 2.4流程图

如图2.1

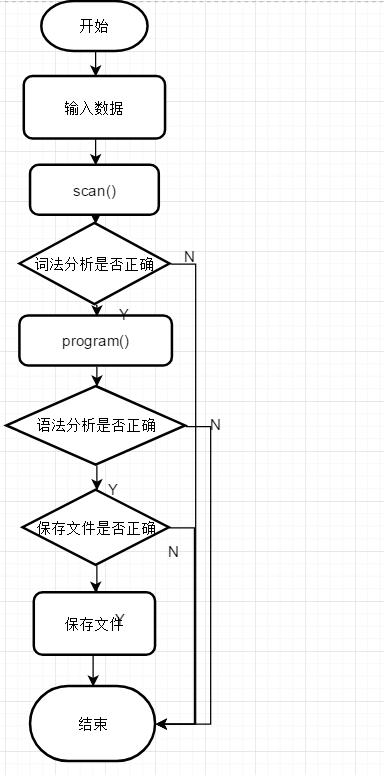


图2.1系统总体流程图

## 3.数据结构和算法详细设计

### 3.1词法分析设计

词法分析包含的数据元素：

表3.1词法分析数据

|  |  |
| --- | --- |
| 输入数据 | 程序数据项定义 |
| 字符串 | 通过区分可以分为标识符，变量，统为ID  或者各种关键字：每种关键字都对应类型 |
| 数字 | 通过区分可分为整型常量INT\_CONST  长整型常量LONG\_CONST |
| 宏定义 | HONG1:#include<stdio.h>  HONG2:#include<stdlib.h>  HONG3:#include<string.h>  HONG4:#include<file.h>  HONG5:#include<math.h> |
| 符号 | 区分各类符号，单目或双目或注释符号 |

词法分析思路：首先将输入的数据代码筛去回车变成一行一行数据存放到一个二维数组s[][]里面，然后在代码块scan()里面在筛去空格变成一个个单词统计输入代码段总体的行数，统计每一行除去空格的字符串的长度，然后通过循环和条件判断语句识别出单词，关键字，符号，数字等等。

词法分析的DFA如下所示，一共分为5中判断：分别能够识别出字母，整型数字，长整型数字，关键字，变量id，符号，注释，宏定义等等。每次从状态0开始，从源程序文件中读取一个字符，可以到达下一个状态，当到达环形的状态（结束状态）时，表示成功的读取到了一个单词，返回单词的编码，保存在全局变量token\_text中。而单词的自身值放在token\_textstring中。

如图3.1

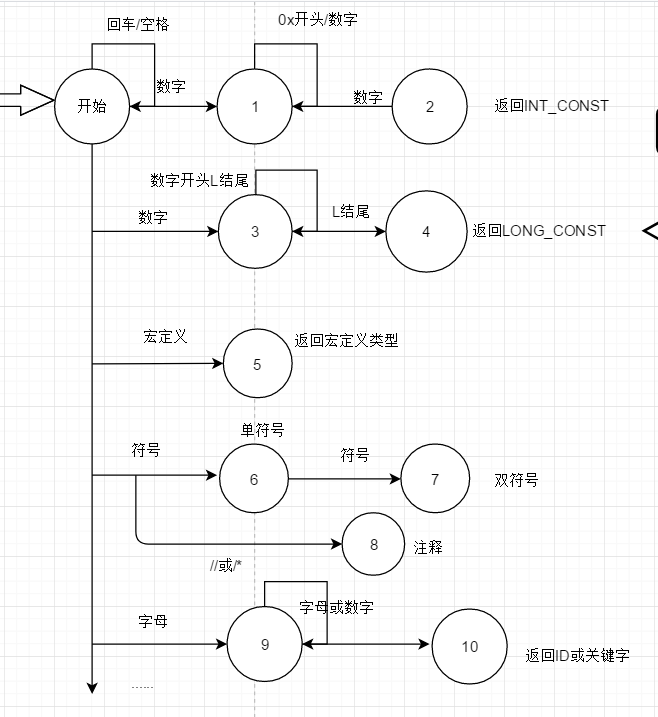


图3.1DFA转换图

### 3.2语法分析设计

语法分析采用递归下降方法的程序结构：在词法分析当中识别出来的单词都通过数组传递到getton\_tokentext()函数里面返回单词相应的编码，并生成一个结构数组，将单词的原有值保存起来，首先进入program()子程序块，这是递归下降程序的开始，通过program程序递归调用其他程序，然后再其他的程序里面也有相应的递归调用，到达功能相符的程序，因为一个源代码开头一定是宏定义，之后或者是外部变量的定义或者是函数类型的定义，可以依照这个规律进行规划子程序的顺序。

如图3.2

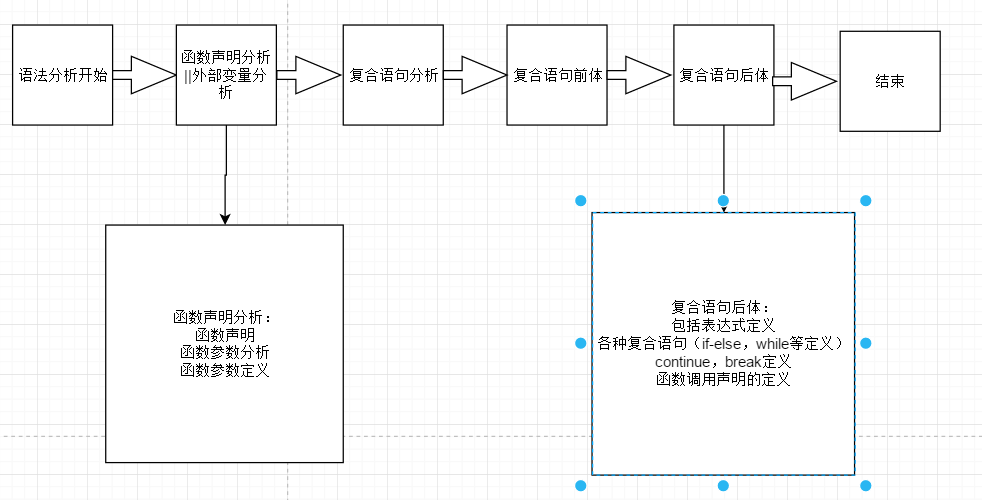


图3.2语法分析设计图

### 3.3保存文件设计

保存文件：依照上述program()代码块生成的分析树先根遍历，其中不同的结点为标记点对后续的符号进行判断，因为一些符号例如’(‘,’{‘等等一些符号并不会写入语法分析书里面，所以要在保存文件的过程中加上。这是一个比较难判断的点，但是在进行词法分析的过程中，将单词的原值保存在了一个结构数组里面，子要将分析树的结点和数组元素对应上，这个功能还是可以实现的。

## **系统实现和测试**

### 4.1词法分析

#### 4.1.1词法分析数据定义

typedef enum

{

ELSE=1,IF,INT,RETURN,VOID,WHILE,FOR,CHAR,FLOAT,LONG,LEQ,REQ,EQ,BREAK,CONTINUE,UNEQ,//16 PLUS,MINUS,TIMES,DIVISION,BIG,LITTLE,ASSIGN,SEMI,COMMA,LP,RP,LMP,//12

RMP,LBP,RBP,ERROR\_TOKEN,IDENT,INT\_CONST,LONG\_CONST,CHAR\_CONST,FLOAT\_CONST,ID,END, //11

HONG1,HONG2,HONG3,HONG4,HONG5//5

}Token\_texttype;

一共44个数据类型，依次对应如下：

else,if,int,return,void,while,for,float,long,”>=”,”<=”,”==”,break,continue,”!=”//16

‘+’,’-’,’\*’,’/’,>’,’<’,’=’, ’;’ , ’,’ , ’(‘, ’)’, ‘[‘//12

‘ ]’, ‘{‘, ‘}’, 未知，整型常数，长整型常数，字符型常数，浮点型常数，标识符，end//11

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<file.h>

#include<math.h>//5

Token结构体的定义：

typedef struct token\_text//token结构体

{

Token\_texttype token\_texttype;//token类型

char token\_textstring[1100];//token串，储存原值

int linenumber;//token行号

}Token;

表4.1词法分析函数及其功能

|  |  |
| --- | --- |
| 函数 | 功能 |
| Token\_texttype gettoken\_texttype(char c[]) | 返回单词编码，方便语法分析使用 |
| void scan() | 扫描代码，将单词分类，语法分析主体 |

### 4.1语法分析

#### 4.2.1语法分析树结点定义

我实现语法分析树的方法是，把每一个语句都当作一个节点来做，这样虽然代码段比较长，但是编写和理解阅读的过程中相对来说比较容易，例如，一个赋值语句，首先生成一个赋值的结点，如果是int型，那么生成一个int结点作为赋值结点的第一个孩子，然后后面的id作为int结点的第一个孩子。

typedef struct TREENODE//树节点结构体孩子表示法

{

TREENODE \* child;//左子节点

TREENODE \* rchilde;//右子节点

int linenumber;//目前分析所在行

Nodekind NODEkind;//节点类型

char NODEstring[1100];//节点类型所代表的字符串，用于语法树打印

}TREENODE,\*tokenTREE;

结点类型有如下表示：

typedef enum Nodekind

{ ints,ids,voids,chars,floats,longs,INT\_CONSTS,LONG\_CONSTS,var,shuzu,hanshu,hancanlist,hancan,fuheyuju,ifs,elses,whiles,returns,breaks,continues,

fuzhi,yunsuan,hong,hong1,hong2,hong3,hong4,hong5,shuzuyuansu,hanshudiaoyong,hanshudiaoyongcanlist,unknown

}NODEkind;//节点种类

注：很多结点类型名称都是汉字的拼音。

#### 4.2.2递归下降算法分析

函数定义：

void gettoken\_text();//得到token

void programerror();//输出错误

void match(Token\_texttype toke);//匹配

TREENODE \* compound\_stmt();//函数声明

void printspace(int n);//打印空格

void printTREE(TREENODE \* t);//打印语法分析树

TREENODE \* NEWNODE(Nodekind kind);//创建新节点

TREENODE \* function\_listformat();//函数参数列表格式

TREENODE \* call(TREENODE \* k);//函数调用

TREENODE \* factor(TREENODE \* k); //带括号的表达式

TREENODE \* term(TREENODE \* k); //乘除表达式

TREENODE \* additive\_expression(TREENODE \* k);//加成的表达式

TREENODE \* simple\_expression(TREENODE \* k);//简单表达式

TREENODE \* variabler(); //变量声明

TREENODE \* expression(); //表达式声明

TREENODE \* expression\_stmt();//表达式声明

TREENODE \* return\_stmt();//返回式声明

TREENODE \* continue\_stmt();//continue声明

TREENODE \* break\_stmt();//break声明

TREENODE \* while\_stmt();//while声明

TREENODE \* if\_stmt();//if声明

TREENODE \* else\_stmt();//else声明

TREENODE \* statement();//复合语句后者种类

TREENODE \* statement\_list();//复合语句体后者

TREENODE \* local\_declaration();//复合语句体前者

TREENODE \* function\_def(TREENODE \* k);//函参

TREENODE \* function\_def\_list(TREENODE \* k);//函参列表

TREENODE \* compound\_stmt();//函数内容，复合语句

TREENODE \* function\_defs();//函数参数

TREENODE \* declaration();//函数声明

TREENODE \* declaration\_list();//多个函数列表

TREENODE \* program();//语法分析

辅助函数功能如表4.1

表4.2辅助函数功能

|  |  |
| --- | --- |
| 函数名称 | 功能 |
| void gettoken\_text() | 得到数据编码，方便语法分析树生成结点 |
| void programerror() | 输出语法分析错误 |
| void match(Token\_texttype toke) | 匹配当前token和得到的token是否一致 |
| void printspace(int n) | 打印空格，使语法分析树比较格式化 |

下面只分析一部分的递归下降算法的传导，详细代码请看附录源代码

表4.3词法分析函数及其功能

|  |  |
| --- | --- |
| 文法 | 分析 |
| program→declaration-list  实现代码：treeNode \* program() | Program(void)函数使用递归向下分析方法直接调用declaration\_list()函数，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| declaration\_list → declaration{ declaration }  实现代码：treenode \* declaration\_list() | declaration\_list(void)函数使用递归向下分析方法直接调用declaration()函数，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| declaration→function\_defs||compound\_stmt()  实现代码：treenode \* declaration() | declarationvoid)函数使用递归向下分析方法直接调用compound\_stmt()函数或者function\_defs，具体调用什么需要进行判断，具体要看代码宏定义之后是外部变量声明还是函数声明 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| compound\_stmt→ local\_declaration()  ||compound\_stmt→statement\_list()  实现代码：treenode \* compound\_stmt() | compound\_stmt()函数使用递归向下分析方法直接调用local\_declaration()函数或者statement\_list，，具体函数是符合语句还是简单的内部语句需要观察花括号的位置，最后返回树节点到上一级的函数调用 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| function\_defs→ function\_def\_list  实现代码：  treenode \* function\_defs() | function\_def\_list()是函数定义结点生成函数。该函数使用递归向下分析方法直接调用function\_def\_list()函数（函数列表定义函数），并返回树节点，并且返回上一级的函数调用 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| function\_def\_list → function\_def  实现代码：  treenode \* function\_def\_list() | function\_def\_list()函数是函数参数列表声明函数，使用递归向下分析方法直接调用function\_def函数，并返回树节点到上以级函数的调用，具体分析是一个函数参数还是多个函数参数。 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| function\_def  实现代码：treenode \* function\_def() | function\_def()函数是识别函数参数类型的函数，例如可以识别出int型，和float型参数，做问函数声明这一类结点的结尾，不再调用其它函数，并返回树节点到上一级函数调用 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| local\_declaration()  实现代码：treenode \* local\_declaration() | local\_declaration()函数定义的是复合语句的前体定义。考虑到复合语句的开始一般是定义变量，所以该函数是分析变量定义的函数，功能可以区分变量的类型，同时作为上级函数递归调用的终点，不再向下递归调用，最后返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| statement\_list() → statement()  实现代码：treenode \* statement\_list() | statement\_list()函数是复合语句的后体功能列表定义函数，因为在变量定义以后，代码将进行一系列复杂的操作，比如各种循环和条件判断，所以该函数是将具体的功能进行识别交给下一级函数。使用递归向下分析方法直接调用statement()函数，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| statement() → if\_stmt()  ||statement() →else\_stmt()  ||statement() →while\_stmt()  ||statement() →return\_stmt()  ||statement() →break\_stmt()  ||statement() →continue\_stmt()  ||statement() →compound\_stmt()  ||statement() →expression\_stmt()  实现代码：treenode \* statement() | statement()函数是复合语句的后体功能定义函数，该函数功能是将遇到的下一个语句进行分类，然后再去调用下一级函数，所以该语句将会在程序中不断的被调用，因为复合语句后体的每一行代码都需要被识别，所以他的下一级函数也可以调用他使用递归向下分析方法直接调用如左的各类函数，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| if\_stmt()→expression()  ||else\_stmt()→statement()  ||while\_stmt()→expression();  ||return\_stmt()→statement()  ||break\_stmt()→statement()  ||continue\_stmt()→statement()  ||compound\_stmt()  ||expression\_stmt() | else\_stmt(),return\_stmt(),break\_stmt(),continue\_stmt()这五个函数分别分析识别if，else，break，continue，return并且返回statement进行下一轮的识别。While\_stmt和if\_stmt函数的下一级是表达式函数，因为if和while后面的括号里面有表达式的判断，之后再返回statement(),最后返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| expression() → simple\_expression(t)  实现代码：treenode \* expression() | expression()函数是表达式定义函数，该函数的功能是首先判断目前到达的表达式是否为赋值语句，如果是返回处理后的结点到上一级。否则使用递归向下分析方法直接调用simple\_expression(t)函数，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| simple\_expression()  → additive\_expression(k)  实现代码：treenode \* simple\_expression() | simple\_expression() 函数是简单表达式定义函数，该函数的功能是识别当前的单词是否符合要求，要求为单目的表示符，如果是那么进行下一级的调用，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| additive\_expression() → term(k)  实现代码：treenode \* additive\_expression() | additive\_expression() 函数实现加减结点的构造功能，调用下一级，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| term() → factor(k)  实现代码：treenode \* term() | term() 函数实现乘除结点的构造功能，调用下一级，并返回树节点 |

|  |  |
| --- | --- |
| 文法 | 分析 |
| factor(k)  实现代码：treenode \* factor() | Factor() 函数实现，首先是识别变量之后的字符元素，并且判定是否是变量的声明又或者是函数的调用，可以调用其他很多结点，例如源代码中要实现调用函数的功能，factor将会去调用call函数，最后返回树节点 |

### 4.3保存文件设计

#### 4.3.1保存文件设计过程数据定义

结构定义：

typedef struct token\_text//token结构体

{

Token\_texttype token\_texttype;//token类型

char token\_textstring[1100];//token串

int linenumber;//token行号

}Token;

int tttt; //tokenlist里面的元素个数

Token tokenlist[1000]; //token表声明

变量定义：

int fff = 0; //tokenlist[]下标

int flag1 = 0;

int flag2 = 0;

int fvoid = 0;

int fbp = 0;

#### 4.3.2保存文件代码分析

表4.4保存文件函数名称及其功能

|  |  |
| --- | --- |
| 函数名称 | int savefile(TREENODE \* T)和void save(FILE \* FP) |
| 具体分析 | 本代码块是由ast语法树和提前保存的单词原值数组一同实现的，首先先根遍历进行判断当前结点是什么，然后每次传递一个节点，结构数组的下表增长一位，因为生成语法分析树的时候并没有将一些代码中含有的特殊符号加进去，所以结构数组的作用是在保存文件的时将这些特殊符号加进去。 |

### 4.4.系统测试

1.实验环境条件：p3 3.2GHz，6G内存：WINDOWS10，Dev .C++

操作指令如表4.5

表4.5具体操作指令

|  |  |
| --- | --- |
| 操作指令 | 功能 |
| 输入代码，回车以ctrl+z结尾 | 读入数据 |
| 若使用代码1，输入1 | 打印出语法分析树 |
| 若使用代码2，输入2，输入文件名A.ref | 以一定的格式保存文件 |
| 退出，输入0 | 退出系统 |

代码1：几乎包含了所有的词法分析和语法分析的需要，但是由于判断过多，我在保存文件时并没有加上那么多的判断，所以只适合打印语法分析树，代码如下

#include<stdio.h>

//int m;

int n;

//long c;

int fun(int a,int b)

{

int d;

c=123L;

d=0x123;

while(m == n)

{

if (a>b)

{n=a;

return n ;

}

else

{

m=(c+d)\*(b+a);

//continue;

return m;

}

}

}

float x;

/\*aa\*/

void main(void)

{

int a; /\*a\*/

a = fun(c,d); //aa

return 0;

}

代码2：代码段简单，和实验要求所给的代码段基本无异，可以打印语法分析树和保存文件，代码如下

#include<stdio.h>

int n;

int fun(int a,int b)

{

int m;

//int n;

if (a>b)

m=a;

else

m=b+a;

return m;

}

float x;

下图所示为代码1进行操作：

词法分析部分：

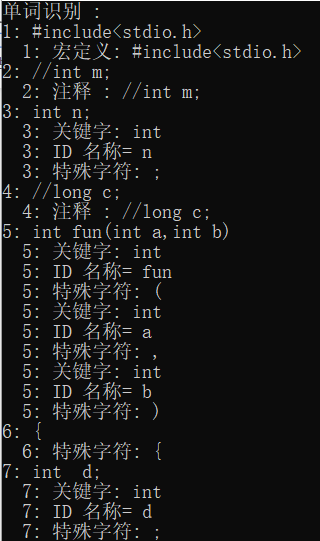


图4.1词法分析截图1

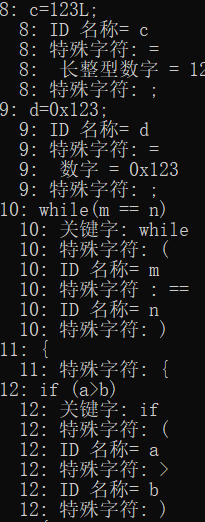


图4.2词法分析截图2

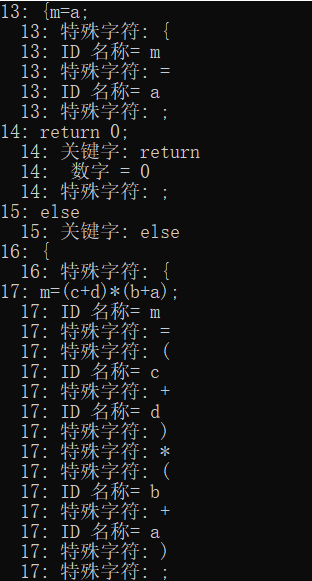


图4.3词法分析截图3

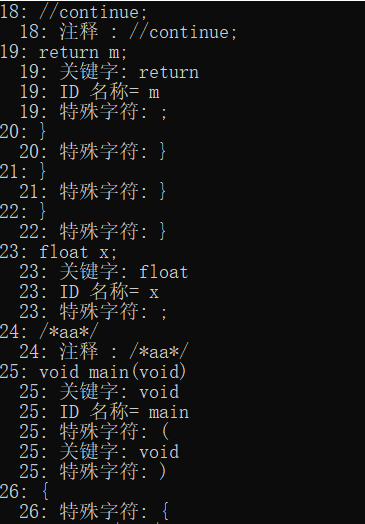


图4.4词法分析截图4



图4.5词法分析截图5

语法分析部分：

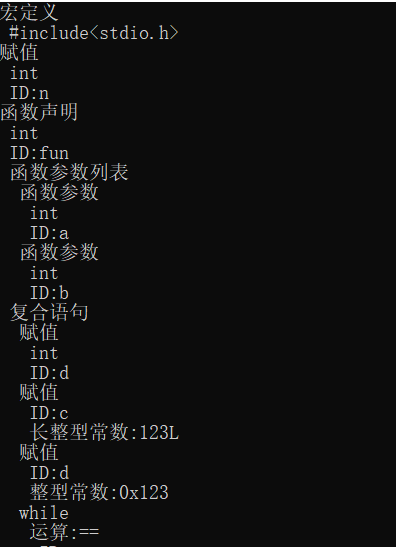


图4.6语法分析截图1

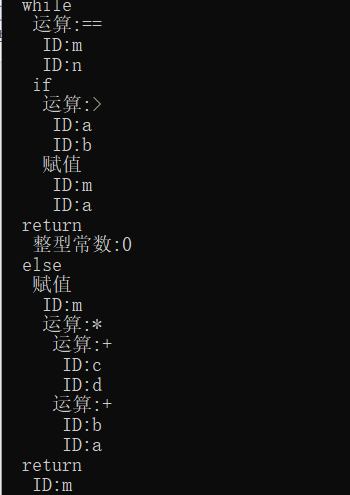


图4.7语法分析截图2

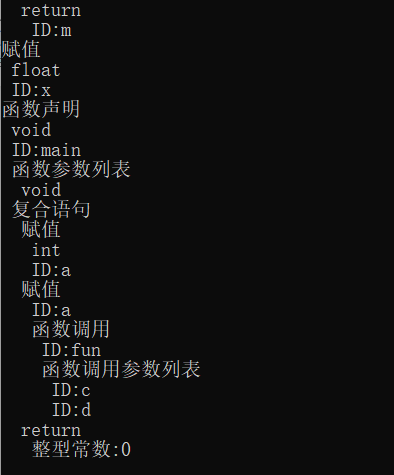


图4.8语法分析截图3

保存文件部分：

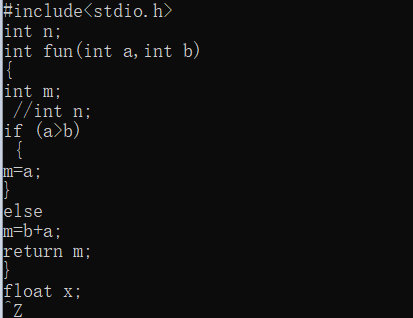


图4.8测试代码输入

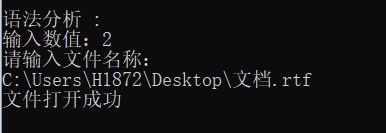


图4.9操作指令截图

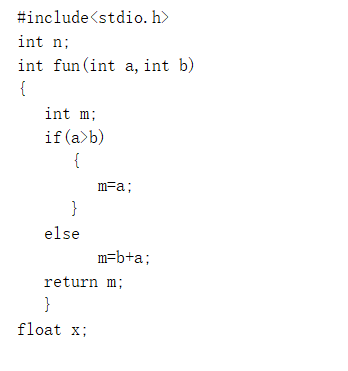
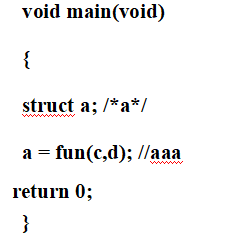
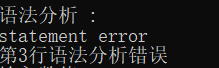


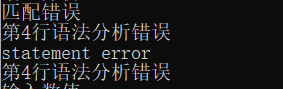
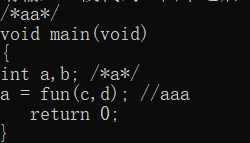
图4.10文件内部截图

出错案例1：第3行出现未规定的词

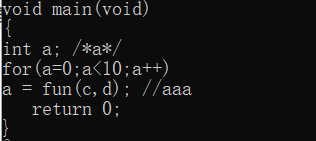


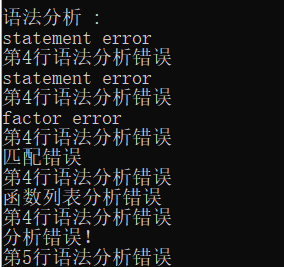


出错案例2：第4行不能生成结点，我规定的必须int 一个变量，如果出现int a，b；程序将会叫停出错。



出错案例3:我写的程序不能识别for循环，出现错误





## 总结与展望

### 5.1全文总结

对自己的工作做个总结，主要工作如下：

（1）第一步，在做课设以前，一直都在抉择应该做那个好，因为是放假期间，所以学习的效率比较低，看第一个课设的实验指导很长时间都没有什么门路，所以直接选择了第二个课设，这份课设网上资源比较少，但是数据结构课上所学习的内容大部分都可以用到，而且开展也比较有规律大体分为两大块。在做之前需要知道数据结构应该怎样去进行构建，才能达到迅速有效的结果。我觉得最重要的就是完成数据结构的搭建工作了。

（2）第二步进行编写，在词法分析部分还算比较正常和简单，只需要识别单词和各种符号即可，但是在语法分析构造语法分析树的过程中，遇到了很多的障碍，这里一方面要考虑语法分析树结构的改变，另一方面要考虑到各种函数功能的关联问题，与梯度下降算法的理解有很大的关联，这需要很强的逻辑思维，耗时时间最长没花费精力也最多

（3）程序最后一部分时根据生成的语法分析树保存文件，这里有种让我从头开始的感觉，因为在生成语法分析树的过程中，并没有将一些特殊的符号写在里面，在节点遍历的过程中并不能找到这些符号，所以又要用到词法分析时保存过的单词的原值，所以各种循环和判断比较繁琐。

### 5.2工作展望

在今后的研究中，围绕着如下几个方面开展工作

（1）对程序的优化，我目前所编写的这个程序有很大的局限性，对一些复杂的语句不能识别出来，比如for，scanf，printf等等，而且代码里面也有很多的bug，不能够适用于所有的代码，一些错误会时常出现，而且我是根据任务书先进行编写的测试代码，这样有一些对照的作用，然后根据测试的代码修改完善程序，所以对测试代码改变太大会出现很多无法识别的状况。

（2）程序中出现了很多不必要的循环，下一步要实现的是减少这个程序的时间复杂度和空间复杂度，正如我所了解到的，梯度下降算法是一些编译器内部的核心算法，如果对程序加以改进和完善，并且缩短识别需要的时间，那么就可以构建出来一个简单的智能编译器了。

## 6.体会

### 6.1总结

本次课程设计是我第一次系统的进行一个比较复杂的程序的编写。当然作为一个正在学习的程序员，这可能仅仅是个开始，但是代码的确是我编写的最长的代码了，可能可以优化的地方还很多，不至于达到2000行以上。

我最有收获的一点就是对C语言以及C语言的数据结构有了更深的体会，并且学习并且了解熟悉了梯度下降算法的主要思想，使我的思维变得更加的严谨，而且在本次课程设计中增加了自己的动手能力，锻炼了构造一个项目的框架方法，能够很好的给出系统的框架，并且能够按照程序流程进行程序的编写。能够运用递归下降的方法进行分析问题，在词法分析中掌握了状态机的转变，同时能够熟练运用状态机进行字符的匹配。在语法分析中，学习到了怎么样通过文法构建代码，以及根据文法编写代码带来的相关问题，如左递归，节点的定义等。

最后，在完成代码的时刻，心里有一种很轻松的感觉，不忘危这么多天的废寝忘食，并且最后回想一下，自己也确实发现了很多的不足，时间的浪费太严重，而且任务完成比较拖沓，通常一个任务要2天才能完成，这也是时间给的过于充分的副作用吧。并且一边写代码一边发现问题并且解决问题的过程还是很值得人来回忆的，并且很有成就感。

### 6.2特色

1. 生成结构数组来保存单词的原值和单词的类型，就可以统一的进行调用并且完成要求。
2. 宏定义结点的增加和注释的判别，程序能够判别行注释和段注释。
3. 使测试代码每一行的语句都生成一棵子树，这样能够统一树的结构，并且能给编写程序的过程带来很大的方便。
4. 保存文件时的单词原值遍历和语法树先根遍历的共同使用。

### 6.3不足

1. 循环结构体运用的过多，使时间复杂度上升。
2. 静态分配数组来存储测试代码的数据，空间复杂度上升。
3. 不能识别很多复杂语句生成语法分析树。
4. 程序还存在一些bug不时弹出，还未找到原因进行改正。

## 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). 清华大学出版社

[3] 百度百科. 抽象语法树

## 附录

### 附录1：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define MAXLENG 10

#define MAXSIZE 2

#define INFEASTABLE -1

#define OVERFLOW -2

#define QMAXSIZE 5 //队列大小

typedef int status; //数据元素类型定义

typedef char TElemType;//定义字符型常量

#define LINE 1024

static char keywords[50][20]={ "char", "continue","break",

"else","for", "float","if","while",

"int","double","case",

"return","void","continue", "struct",

"switch","default"};

static char headfile[50][30]={ "#include<stdio.h>","#include<stdlib.h>","#include<string.h>","#includ<file.h>","#include<math.h>"};

typedef enum

{

ELSE=1,IF,INT,RETURN,VOID,WHILE,FOR,CHAR,FLOAT,LONG,LEQ,REQ,EQ,BREAK,CONTINUE,UNEQ,//16

PLUS,MINUS,TIMES,DIVISION,BIG,LITTLE,ASSIGN,SEMI,COMMA,LP,RP,LMP,//12

RMP,LBP,RBP,ERROR\_TOKEN,IDENT,INT\_CONST,LONG\_CONST,CHAR\_CONST,FLOAT\_CONST,ID,END, //11

HONG1,HONG2,HONG3,HONG4,HONG5//5

}Token\_texttype;

typedef enum Nodekind

{

ints,ids,voids,chars,floats,longs,INT\_CONSTS,LONG\_CONSTS,variable,shuzu,hanshu,hancanlist,hancan,fuheyuju,ifs,elses,whiles,returns,breaks,continues,

fuzhi,yunsuan,hong,hong1,hong2,hong3,hong4,hong5,shuzuyuansu,hanshudiaoyong,hanshudiaoyongcanlist,unknown

}NODEkind;//节点种类

typedef enum Expressiontype

{

Void,

Int,

Char,

Float

}expressiontype;//表达式种类

typedef struct TREENODE//树节点结构体孩子表示法

{

TREENODE \* child;//左子节点

TREENODE \* rchilde;//右子节点

int linenumber;//所在行

Nodekind NODEkind;//节点类型

char NODEstring[1100];//节点类型所代表的字符串，用于语法树打印

}TREENODE,\*tokenTREE;

typedef struct token\_text//token结构体

{

Token\_texttype token\_texttype;//token类型

char token\_textstring[1100];//token串

int linenumber;//token行号

}Token;

Token\_texttype gettoken\_texttype(char c[]);

void scan(); //扫描，词法识别

char s[1005][1005];//文本

char p[1005][1005];//临时替换数组

char t[1005];//当前token\_text串

char tk[1005];

char linshi[1100];//临时数组

int line; //文本行数

int zs; //注释标记

int tttt; //标记，结构数组个数

Token tokenlist[1000]; //token表声明

Token ctoken;//当前token

Token ltoken;//上一个token

int xb=0;//token下标

int blank=0;//先行空格

void gettoken\_text();//得到token

void programerror();//输出错误

void match(Token\_texttype toke);//匹配

TREENODE \* compound\_stmt();//函数声明

void printspace(int n);//打印空格

void printTREE(TREENODE \* t);//打印语法分析树

TREENODE \* NEWNODE(Nodekind kind);//创建新节点

TREENODE \* function\_listformat();//函数参数列表格式

TREENODE \* call(TREENODE \* k);//函数调用

TREENODE \* factor(TREENODE \* k); //带括号的表达式

TREENODE \* term(TREENODE \* k); //乘除表达式

TREENODE \* additive\_expression(TREENODE \* k);//加成的表达式

TREENODE \* simple\_expression(TREENODE \* k);//简单表达式

TREENODE \* variabler(); //变量声明

TREENODE \* expression(); //表达式声明

TREENODE \* expression\_stmt();//表达式声明

TREENODE \* return\_stmt();//返回式声明

TREENODE \* continue\_stmt();//continue声明

TREENODE \* break\_stmt();//break声明

TREENODE \* while\_stmt();//while声明

TREENODE \* if\_stmt();//if声明

TREENODE \* else\_stmt();//else声明

TREENODE \* statement();//复合语句后者种类

TREENODE \* statement\_list();//复合语句体后者

TREENODE \* local\_declaration();//复合语句体前者

TREENODE \* function\_def(TREENODE \* k);//函参

TREENODE \* function\_def\_list(TREENODE \* k);//函参列表

TREENODE \* compound\_stmt();//函数内容，复合语句

TREENODE \* function\_defs();//函数参数

TREENODE \* declaration();//函数声明

TREENODE \* declaration\_list();//多个函数列表

TREENODE \* program();//语法分析

void savefile(TREENODE \* T);//,FILE \*FP);//保存文件

void save(FILE \* FP);

int main()

{

/\*FILE \*FP;

int i = 0,j = 0;

char w,c;

printf("请输入将要读取的文件名，以.txt结尾：\n");

char file[30];

scanf("%s",file);

if ((FP=fopen(file,"rb")) == NULL)

printf("File open error\n ");

else

printf("文件读取成功\n");

while((c = fgetc(FP)) == 1)

{

if(c == '\n')

{

j = 0;

i++;

}

else

s[i][j++] = c;

line = i;

if(c == '#')

break;

}

fclose(FP); \*/

printf("请输入一段代码，回车之后ctrl+z结束：\n");

int i=0;

int change;

line=0;

while(gets(s[line++]))

{}

printf("单词识别 :\n");

scan();

// zhushi();

printf("\n语法分析 :\n");

TREENODE \*t = program(); //生成ast树头结点

TREENODE \*p;

p = t;

printf("输入数值：");

scanf("%d",&change);

getchar();

if(change == 1)

printTREE(t); //打印ast语法树

else if(change == 2)

{

FILE \*FP;

printf("请输入文件名称：\n");

char file[30];

scanf("%s",file);

if ((FP=fopen(file,"w")) == NULL)

printf("文件打开失败\n ");

else

printf("文件打开成功\n");

savefile(p);

save(FP);

fclose(FP);

}

else

{

getchar();

return 0;

}

getchar();

return 0;

}

Token\_texttype gettoken\_texttype(char c[])

{

Token\_texttype to;

if(!strcmp(c,"else"))

{

to=ELSE;

}

if(!strcmp(c,"break"))

{

to=BREAK;

}

if(!strcmp(c,"continue"))

{

to=CONTINUE;

}

else if(!strcmp(c,"if"))

{

to=IF;

}

else if(!strcmp(c,"return"))

{

to=RETURN;

}

else if(!strcmp(c,"int"))

{

to=INT;

}

else if(!strcmp(c,"char"))

{

to=CHAR;

}

else if(!strcmp(c,"float"))

{

to=FLOAT;

}

else if(!strcmp(c,"long"))

{

to=LONG;

}

else if(!strcmp(c,"void"))

{

to=VOID;

}

else if(!strcmp(c,"while"))

{

to=WHILE;

}

else if(!strcmp(c,"+"))

{

to=PLUS;

}

else if(!strcmp(c,"-"))

{

to=MINUS;

}

else if(!strcmp(c,"\*"))

{

to=TIMES;

}

else if(!strcmp(c,"/"))

{

to=DIVISION;

}

else if(!strcmp(c,"<"))

{

to=LITTLE;

}

else if(!strcmp(c,"<="))

{

to=LEQ;

}

else if(!strcmp(c,"#include<stdio.h>"))

{

to = HONG1;

}

else if(!strcmp(c,"#include<stdlib.h>"))

{

to = HONG1;

}

else if(!strcmp(c,"#include<string.h>"))

{

to = HONG1;

}

else if(!strcmp(c,"#include<file.h>"))

{

to = HONG1;

}

else if(!strcmp(c,"#include<math.h>"))

{

to = HONG1;

}

else if(!strcmp(c,">"))

{

to=BIG;

}

else if(!strcmp(c,">="))

{

to=REQ;

}

else if(!strcmp(c,"=="))

{

to=EQ;

}

else if(!strcmp(c,"!="))

{

to=UNEQ;

}

else if(!strcmp(c,"="))

{

to=ASSIGN;

}

else if(!strcmp(c,";"))

{

to=SEMI;

}

else if(!strcmp(c,","))

{

to=COMMA;

}

else if(!strcmp(c,"("))

{

to=LP;

}

else if(!strcmp(c,")"))

{

to=RP;

}

else if(!strcmp(c,"["))

{

to = LMP;

}

else if(!strcmp(c,"]"))

{

to=RMP;

}

else if(!strcmp(c,"{"))

{

to=LBP;

}

else if(!strcmp(c,"}"))

{

to=RBP;

}

else if(!strcmp(c,"INT\_CONST1"))

{

to=INT\_CONST;

}

else if(!strcmp(c,"LONG\_CONST1"))

{

to=LONG\_CONST;

}

else if(!strcmp(c,"ID"))

{

to=ID;

}

else if(!strcmp(c,"end"))

{

to=END;

}

return to;

}

void scan()

{

zs = 0;

char uu[3];

int tt = 0;//tokenlist的下标

int i,j,m,l,n,b,d;//下标

int f,flag;//标记

int len;//当前行长度

char keywords[50][20]={ "char", "continue","break",

"else","for", "float","if","while",

"int","long","double","case",

"return","void","continue", "struct",

"switch","default"};//关键字

char headfile[50][30]={ "#include<stdio.h>","#include<stdlib.h>","#include<string.h>","#includ<file.h>","#include<math.h>"};

char compare2[10][1005]={"<=",">=","==","!=","--","++","&&","||"};//双目比较符

char compare1[1005]={'+','-','\*','/','>','<','=',';',':',',','(','%',')','[',']','{','}','!'};//单目符号

//zs=0;注释判断标志

for(i=0;i<line-1;i++)

{

len = strlen(s[i]);

printf("%d: ",i+1);

printf("%s\n",s[i]);

if(len==0)//空行

{

continue;

}

for(j=0;j<len;j++)

{

if(s[i][j]==' ')//空格

{

continue;

}

if(s[i][j] == '#')

{

l = j;

d = 0;

for(;l<len;l++)

{

tk[d++] = s[i][l];

}

tk[d] = '\0';

for(b=0;b<5;b++)

{

if(!strcmp(tk,headfile[b]));

{

printf(" %d: 宏定义: %s\n",i+1,tk);

tokenlist[tt].token\_texttype = gettoken\_texttype(tk);

strcpy(tokenlist[tt].token\_textstring,tk);

tokenlist[tt].linenumber=i+1;

tt++;

j = l-1;

break;}

}

}

else if(s[i][j]<='9'&&s[i][j]>='0')//数字

{

l = j;

m = 0;

if(s[i][l] == '0')

{

t[m++]=s[i][l++];

if(s[i][l] == 'x')

{

t[m++] = s[i][l++];

}

}

while(s[i][l]<='9'&&s[i][l]>='0')

{

t[m++] = s[i][l++];

}

if(s[i][l] == 'L')

{

t[m++] = s[i][l++];

t[m]='\0';

printf(" %d: 长整型数字 = %s\n",i+1,t);

strcpy(linshi,"LONG\_CONST1");

tokenlist[tt].token\_texttype = gettoken\_texttype(linshi);

strcpy(tokenlist[tt].token\_textstring,t);

tokenlist[tt].linenumber=i+1;

tt++;

j=l-1;

}

else

{

t[m]='\0';

printf(" %d: 整型数字 = %s\n",i+1,t);

strcpy(linshi,"INT\_CONST1");

tokenlist[tt].token\_texttype = gettoken\_texttype(linshi);

strcpy(tokenlist[tt].token\_textstring,t);

tokenlist[tt].linenumber=i+1;

tt++;

j=l-1;

}

}

else if((s[i][j]<='z'&&s[i][j]>='a')||(s[i][j]<='Z'&&s[i][j]>='A'))//字母

{

l=j;

m=0;

while((s[i][l]<='z'&&s[i][l]>='a')||(s[i][l]<='Z'&&s[i][l]>='A'))

{

t[m++]=s[i][l++];

}

t[m]='\0';

f=1;//标记

for(n = 0;n < 20;n++)

{

if(!strcmp(t,keywords[n]))

{

f=0;

printf(" %d: 关键字: %s\n",i+1,t);

tokenlist[tt].token\_texttype = gettoken\_texttype(t);

strcpy(tokenlist[tt].token\_textstring,t);

tokenlist[tt].linenumber=i+1;

tt++;

break;

}

}

if(f)

{

printf(" %d: ID 名称= %s\n",i+1,t);

strcpy(linshi,"ID");

tokenlist[tt].token\_texttype = gettoken\_texttype(linshi);

strcpy(tokenlist[tt].token\_textstring,t);

tokenlist[tt].linenumber=i+1;

tt++;

}

j=l-1;

}

else//符号

{

l=j;

n=0;

while(1)

{

flag=0;//标记

for(m=0;m<18;m++)

{

if(s[i][l]==compare1[m])

{

flag=1;

t[n++]=compare1[m];

l++;

break;

}

else if(s[i][l] == '&')

{

flag = 1;

t[n++]='&';

l++;

break;

}

else if(s[i][l]=='|')

{

flag = 1;

t[n++]='|';

l++;

break;

}

}

if(!flag)

{

break;

}

}

t[n]='\0';

b=l;

if(!strcmp(t,"/\*"))//注释开始

{

zs=1;

uu[0]='/';

uu[1]='\*';

uu[2]='\0';

printf(" %d: 注释 : %s",i+1,uu);

while(zs)

{

if(s[i][b]=='\*'&&s[i][b+1]=='/')

{

uu[0]='\*';

uu[1]='/';

uu[2]='\0';

zs = 0;

printf("%s\n",uu);

}

else

{

printf("%c",s[i][b++]);

}

}

j=b+1;

}

else if(!strcmp(t,"//"))

{

zs=1;

uu[0] = '/';

uu[1] = '/';

uu[2] = '\0';

printf(" %d: 注释 : %s",i+1,uu);

for(b=l;b<len;b++)

{

printf("%c",s[i][b]);

}

printf("\n");

j = 0;

break;

}

else

{

flag=1;

for(m=0;m<=5;m++)//双目比较符

{

if(!strcmp(t,compare2[m]))

{

tokenlist[tt].token\_texttype=gettoken\_texttype(t);

strcpy(tokenlist[tt].token\_textstring,t);

tokenlist[tt].linenumber=i+1;

printf(" %d: 特殊字符 : %s\n",i+1,t);

flag=0;

tt++;

break;

}

}

if(flag)//单目符号

{

for(m=0;m<n;m++)

{

linshi[0]=t[m];

linshi[1]='\0';

tokenlist[tt].token\_texttype=gettoken\_texttype(linshi);

strcpy(tokenlist[tt].token\_textstring,linshi);

tokenlist[tt].linenumber=i+1;

printf(" %d: 特殊字符: %c\n",i+1,t[m]);//一个一个输出

tt++;

}

}

j=l-1;

}

}

}

}

tokenlist[tt].linenumber = i;

tokenlist[tt].token\_texttype = END; //////////

strcpy(tokenlist[tt].token\_textstring,"endfile"); /////////////////////

tt++;

tttt = tt - 1;

}

TREENODE \* program()//语法分析，递归 下降 program的开始

{

TREENODE \* t; //生成结点

gettoken\_text(); //得到token结点的数据

t = declaration\_list(); //分析 函数列表

if(ctoken.token\_texttype != END)//一段函数的分析完成 end

{

printf("分析错误！\n");

programerror();

}

return t;

}

void gettoken\_text()//得到token\_text

{

ltoken = ctoken;

ctoken=tokenlist[xb++];

}

void match(Token\_texttype toke)//匹配

{

if(ctoken.token\_texttype == toke)

{

gettoken\_text();

}

else

{

printf("匹配错误\n");

programerror();

}

}

void programerror()//语法程序分析错误

{

printf("第%d行语法分析错误\n",ctoken.linenumber);

}

TREENODE \* NEWNODE(Nodekind kind)//创建新节点

{

TREENODE \* p = (TREENODE \*)malloc(sizeof(TREENODE)); //创建结点过程分配结点的存储空间

if(p == NULL)

{

printf("创建新结点失败!\n"); //创建失败

}

else

{

p->child = NULL; //孩子初始为空

p->rchilde = NULL; //右孩子初始为空

p->linenumber = ctoken.linenumber; //行数右结点的行数

p->NODEkind = kind; //结点类型

strcpy(p->NODEstring,ctoken.token\_textstring); //结点串

//printf("结点字符串：%s\n",p->NODEstring);

}

return p;

}

TREENODE \* declaration\_list()//多个函数列表

{

//TREEBODE \* T;

TREENODE \* t;

t = declaration();

TREENODE \* p;

p = t; //后面有用

while(ctoken.token\_texttype == HONG1||ctoken.token\_texttype == HONG2||ctoken.token\_texttype == HONG3||ctoken.token\_texttype == HONG4||ctoken.token\_texttype == HONG5)

{

TREENODE \* q;

q = declaration();

if(q!=NULL)

{

if(t==NULL)

{

t = q;

p = q;

}

else

{

p->rchilde=q;

p=q;

}

}

}

while(ctoken.token\_texttype!=INT&&ctoken.token\_texttype!=VOID&&ctoken.token\_texttype!=END&&ctoken.token\_texttype != CHAR&&ctoken.token\_texttype != FLOAT&&ctoken.token\_texttype != LONG&&ctoken.token\_texttype != ID) //end为结束标志符

{

printf("函数列表分析错误\n");

programerror();

gettoken\_text();

if(ctoken.token\_texttype==END) //end为结束符

{

break;

}

}

while(ctoken.token\_texttype==INT||ctoken.token\_texttype==VOID||ctoken.token\_texttype==CHAR||ctoken.token\_texttype==FLOAT||ctoken.token\_texttype==LONG)

{

TREENODE \* q;

q = declaration();

if(q!=NULL)

{

if(t==NULL)

{

t = q;

p = q;

}

else

{

p->rchilde=q;

p=q;

}

}

}

return t;

}

TREENODE \* declaration()//函数声明

{

TREENODE \* t;

TREENODE \* p;

TREENODE \* q;

TREENODE \* s;

Nodekind c;

if(ctoken.token\_texttype==INT)//类型匹配

{

p = NEWNODE(ints);

match(INT);

c = ints;

}

else if(ctoken.token\_texttype==VOID)

{

p=NEWNODE(voids);

match(VOID);

c = voids;

}

else if(ctoken.token\_texttype==LONG)

{

p=NEWNODE(longs);

match(LONG);

c = longs;

}

else if(ctoken.token\_texttype == CHAR)

{

p = NEWNODE(chars);

match(CHAR);

c = chars;

}

else if(ctoken.token\_texttype == FLOAT)

{

p =NEWNODE(floats);

match(FLOAT);

c = floats;

}

else if(ctoken.token\_texttype == HONG1||ctoken.token\_texttype == HONG2||ctoken.token\_texttype == HONG3||ctoken.token\_texttype == HONG4||ctoken.token\_texttype == HONG5)

{

if(ctoken.token\_texttype == HONG1)

{

//printf("#include<stdio.h>\n");

p =NEWNODE(hong1);

match(HONG1);

c = ints;

}

else if(ctoken.token\_texttype == HONG2)

{

//printf("#include<stdlib.h>\n");

p =NEWNODE(hong2);

match(HONG2);

c = hong2;

}

else if(ctoken.token\_texttype == HONG3)

{

//printf("#include<string.h>\n");

p =NEWNODE(hong3);

match(HONG3);

c = hong3;

}

else if(ctoken.token\_texttype == HONG4)

{

//printf("#include<file.h>\n");

p =NEWNODE(hong4);

match(HONG4);

c = hong4;

}

else if(ctoken.token\_texttype == HONG5)

{

//printf("#include<math.h>\n");

p =NEWNODE(hong5);

match(HONG5);

c = hong5;

}

t = NEWNODE(hong);

t->child = p;

return t;

}

else

{

printf("函数声明错误\n");

programerror();

}

if(p!=NULL&&ctoken.token\_texttype == ID)

{

q=NEWNODE(ids);

match(ID);

if(ctoken.token\_texttype==LP)//函数声明

{

t=NEWNODE(hanshu);

t->child = p;

p->rchilde = q;

match(LP);

q->rchilde=function\_defs();

match(RP);

if(q->rchilde!=NULL)

(q->rchilde)->rchilde=compound\_stmt();

else

q->child=compound\_stmt();

}

else if(ctoken.token\_texttype==LMP)//数组声明

{

t=NEWNODE(variable);

match(LMP);

TREENODE \* m;

m=NEWNODE(shuzu);

s=NEWNODE(INT\_CONSTS);

match(INT\_CONST);

t->child=p;

if(p!=NULL)

p->rchilde=m;

m->child=q;

if(q!=NULL)

q->rchilde=s;

match(RMP);

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

}

else if(ctoken.token\_texttype==SEMI) //变量定义声明

{

t = NEWNODE(variable);

t->child=p;

if(p!=NULL)

p->rchilde=q;

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

}

else

{

printf("declaration error\n");

programerror();

}

}

else

{

printf("函数定义错误\n");

programerror();

}

return t;

}

TREENODE \* function\_defs()//函数参数

{

TREENODE \* t;

t=NEWNODE(hancanlist);

TREENODE \* p;

p=NULL;//初始为空

if(ctoken.token\_texttype==VOID)//函参是void或void id{[]}

{

p = NEWNODE(voids);

match(VOID);

if(ctoken.token\_texttype==RP)//(void)类型

{

t->child = p; //如果是右括号，那么void就是括号内部的参数

}

else

{

t->child = function\_def\_list(p);//(void id{[]})类型 函数参数列表

}

}

else if(ctoken.token\_texttype==INT)//(int id{[]})类型

{

p = NEWNODE(ints);

match(INT);

t->child=function\_def\_list(p);

}

else if(ctoken.token\_texttype==CHAR)

{

p = NEWNODE(chars);

match(CHAR);

t->child=function\_def\_list(p);

}

else if(ctoken.token\_texttype==FLOAT)

{

p = NEWNODE(floats);

match(FLOAT);

t->child = function\_def\_list(p);

}

else if(ctoken.token\_texttype==LONG)

{

p = NEWNODE(longs);

match(LONG);

t->child = function\_def\_list(p);

}

return t;

}

TREENODE \* compound\_stmt()//函数内容，复合语句

{

TREENODE \* t;

if(ctoken.token\_texttype == ELSE)

{

t = statement\_list();

goto end1;

}

else if(ctoken.token\_texttype==WHILE)

{

t = statement\_list();

goto end1;

}

else if(ctoken.token\_texttype==RETURN)

{

t = statement\_list();

goto end1;

}

else if(ctoken.token\_texttype==BREAK)

{

t = statement\_list();

goto end1;

}

else if(ctoken.token\_texttype==CONTINUE)

{

t = statement\_list();

goto end1;

}

else

{

t = NEWNODE(fuheyuju);

if(ctoken.token\_texttype == LBP)

match(LBP);

t->child=local\_declaration();

if(t->child!=NULL)

(t->child)->rchilde=statement\_list(); //语句列表

else

t->child=statement\_list();

while(ctoken.token\_texttype == RBP)

match(RBP);

return t;

}

end1:

return t;

}

TREENODE \* function\_def\_list(TREENODE \* k)//函参列表

{

TREENODE \* t=function\_def(k);//只有一个函参的情况

TREENODE \* p=t;

k=NULL;

while(ctoken.token\_texttype==COMMA)//多个函参

{

TREENODE \* q;

match(COMMA);

q = function\_def(k);

if(q!=NULL)

{

if(t==NULL)

{

t = q;

p = q;

}

else

{

p->rchilde = q;

p = q;

}

}

}

return t;

}

TREENODE \* function\_def(TREENODE \* k)//函参

{

TREENODE \* t=NEWNODE(hancan); //生成函数参数结点

TREENODE \* p=NULL;//t的child

TREENODE \* q=NULL;//p的rchilde

if(k == NULL&&ctoken.token\_texttype==INT)//解决viod ID的问题

{

p = NEWNODE(ints); //生成以int为根结点的树结点

match(INT);

}

else if(k == NULL&&ctoken.token\_texttype==CHAR)

{

p = NEWNODE(chars);

match(CHAR);

}

else if(k == NULL&&ctoken.token\_texttype==LONG)

{

p = NEWNODE(longs);

match(LONG);

}

else if(k == NULL&&ctoken.token\_texttype == FLOAT)

{

p = NEWNODE(floats);

match(FLOAT);

}

else if(k!=NULL)

{

p = k;

}

if(p!=NULL)

{

t->child=p;

if(ctoken.token\_texttype==ID)

{

q=NEWNODE(ids);

p->rchilde=q;

match(ID);

}

else

{

printf("函数参数分析错误\n");

programerror();

}

if(ctoken.token\_texttype==LMP&&q!=NULL)

{

match(LMP);

q->rchilde=NEWNODE(INT\_CONSTS);

match(INT\_CONST);

match(RMP);

}

else

{

return t;

}

}

else

{

printf("函数参数分析错误\n");

programerror();

}

return t;

}

TREENODE \* local\_declaration()//复合语句体前者

{

TREENODE \* t=NULL;

TREENODE \* p=NULL;

TREENODE \* q=NULL;

TREENODE \* q1=NULL;

TREENODE \* q2=NULL;

TREENODE \* q3=NULL;

TREENODE \* q4 =NULL;

while(ctoken.token\_texttype==INT||ctoken.token\_texttype==VOID||ctoken.token\_texttype==LONG||ctoken.token\_texttype==FLOAT||ctoken.token\_texttype==CHAR)//有变量声明

{

q=NEWNODE(variable);//变量声明

if(ctoken.token\_texttype==INT)//int声明

{

q1=NEWNODE(ints);

q->child=q1;

match(INT);

}

else if(ctoken.token\_texttype==LONG)//long声明

{

q1=NEWNODE(longs);

q->child=q1;

match(LONG);

}

else if(ctoken.token\_texttype==CHAR)//char声明

{

q1=NEWNODE(chars);

q->child=q1;

match(CHAR);

}

else if(ctoken.token\_texttype==FLOAT)//float声明

{

q1=NEWNODE(floats);

q->child=q1;

match(FLOAT);

}

else if(ctoken.token\_texttype==VOID)//void声明

{

q1=NEWNODE(voids);

q->child=q1;

match(VOID);

}

if(q!=NULL&&ctoken.token\_texttype==ID)

{

q2=NEWNODE(ids);

q1->rchilde=q2;

// q1->child=q2;

match(ID);

if(ctoken.token\_texttype==LMP)

{

q3=NEWNODE(shuzu);

q3->rchilde = q2;

match(LMP);

q4 = NEWNODE(INT\_CONSTS);

match(INT\_CONST);

match(RMP);

q1->rchilde = q3;

q2->rchilde = q4;

}

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

}

else

{

printf("local\_declaration error\n");

programerror();

}

if(q!=NULL)

{

if(t==NULL)

{

t=q;

t=q;

}

else

{

p->rchilde=q;

p=q;

}

}

}

return t;

}

TREENODE \* statement\_list()//复合语句体后者

{

TREENODE \* t=NULL;

t=statement();

TREENODE \* p=t;

while(ctoken.token\_texttype==IF||ctoken.token\_texttype==LP||

ctoken.token\_texttype==ID||ctoken.token\_texttype==WHILE||

ctoken.token\_texttype==RETURN||ctoken.token\_texttype==SEMI||

ctoken.token\_texttype==BREAK||ctoken.token\_texttype==CONTINUE||

ctoken.token\_texttype==LBP||ctoken.token\_texttype==ELSE||ctoken.token\_texttype==INT\_CONST||ctoken.token\_texttype==LONG\_CONST||ctoken.token\_texttype == FOR) // 判断是否为复合语句和条件语句

{

TREENODE \* q;

q = statement();

if(q!=NULL)

{

if(t==NULL)

{

t = q;

p = q;

}

else

{

p->rchilde=q;

p=q;

}

}

}

return t;

}

TREENODE \* statement()//复合语句后者种类

{

TREENODE \* t=NULL;

if(ctoken.token\_texttype==IF)

{

t = if\_stmt();

}

else if(ctoken.token\_texttype==ELSE)

{

t = else\_stmt();

}

else if(ctoken.token\_texttype==WHILE)

{

t = while\_stmt();

}

else if(ctoken.token\_texttype==RETURN)

{

t = return\_stmt();

}

else if(ctoken.token\_texttype==BREAK)

{

t = break\_stmt();

}

else if(ctoken.token\_texttype==CONTINUE)

{

t = continue\_stmt();

}

else if(ctoken.token\_texttype==RBP)

{

t = compound\_stmt();

}

else if(ctoken.token\_texttype==ID)

{

t = expression\_stmt();

}

/\*else if(ctoken.token\_texttype == FOR)

{

t = for\_stmt();

}\*/

else

{

printf("statement error\n");

programerror();

gettoken\_text();

}

return t;

}

TREENODE \* if\_stmt()//if声明

{

TREENODE \* t=NEWNODE(ifs);

match(IF);

match(LP);

if(t!=NULL)

{

t->child=expression();

}

match(RP);

if(ctoken.token\_texttype == LBP)

{

match(LBP);

}

if(t->child!=NULL)

(t->child)->rchilde=statement();

return t;

}

TREENODE \* else\_stmt()

{

match(ELSE);

if(ctoken.token\_texttype == LBP)

{

match(LBP);

}

TREENODE \* t = NEWNODE(elses);

if(t!=NULL)

{

t->child=statement();

}

return t;

}

TREENODE \* while\_stmt()//while声明

{

TREENODE \* t=NEWNODE(whiles);

match(WHILE);

match(LP);

if(t!=NULL)

{

t->child=expression();

}

match(RP);

match(LBP);

if(t!=NULL)

{

(t->child)->rchilde=statement();

}

return t;

}

TREENODE \* return\_stmt()//返回式声明

{

TREENODE \* t=NEWNODE(returns);

match(RETURN);

if(ctoken.token\_texttype==SEMI)

{

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

return t;

}

else

{

if(t!=NULL)

{

t->child=expression();

}

}

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

return t;

}

TREENODE \* break\_stmt()

{

TREENODE \* t = NEWNODE(breaks);

match(BREAK);

if(ctoken.token\_texttype==SEMI)

{

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

return t;

}

}

TREENODE \* continue\_stmt()

{

TREENODE \* t = NEWNODE(continues);

match(CONTINUE);

if(ctoken.token\_texttype==SEMI)

{

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

return t;

}

}

TREENODE \* expression\_stmt()//表达式声明

{

TREENODE \* t=NULL;

if(ctoken.token\_texttype==SEMI)

{

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

}

else

{

t=expression();

match(SEMI);

if(ctoken.token\_texttype==RBP)

match(RBP);

}

return t;

}

TREENODE \* expression()//表达式

{

TREENODE \* t=variabler(); //定义结点t是一个变量声明的结点

if(t == NULL)

{

t=simple\_expression(t);

}

else

{

TREENODE \* p=NULL;

if(ctoken.token\_texttype == ASSIGN)

{

p=NEWNODE(fuzhi);

match(ASSIGN); //判断条件语句里的赋值语句

p->child=t;

(p->child)->rchilde=expression();

return p;

}

else

{

t=simple\_expression(t);

}

}

return t;

}

TREENODE \* variabler()//变量声明

{

TREENODE \* t=NULL;

TREENODE \* p=NULL;

TREENODE \* q=NULL;

if(ctoken.token\_texttype==ID)

{

p=NEWNODE(ids); //新建一个id结点

match(ID);

if(ctoken.token\_texttype==LMP) //数组的定义

{

match(LMP);

if(ctoken.token\_texttype==INT\_CONST)

{

q = NEWNODE(INT\_CONSTS);

match(INT\_CONST);

match(RMP);

match(SEMI);

}

else

q = expression(); //表达式函数调用 赋给一个实际的值

t=NEWNODE(shuzuyuansu); //生成一个数组元素结点

t->child=p;

p->rchilde=q;

}

else

{

t=p;

}

}

return t;

}

TREENODE \* simple\_expression(TREENODE \* k)//简单表达式

{

TREENODE \* t=additive\_expression(k);

k=NULL;

if(ctoken.token\_texttype==LEQ||ctoken.token\_texttype==LITTLE||

ctoken.token\_texttype==BIG||ctoken.token\_texttype==REQ||

ctoken.token\_texttype==EQ||ctoken.token\_texttype==UNEQ)

{

TREENODE \* q=NEWNODE(yunsuan);

match(ctoken.token\_texttype);

q->child=t;

(q->child)->rchilde=additive\_expression(k);

return q;

}

return t;

}

TREENODE \* additive\_expression(TREENODE \* k)//加成的表达式

{

TREENODE \* t=term(k);

k=NULL;

while(ctoken.token\_texttype==PLUS||ctoken.token\_texttype==MINUS)

{

TREENODE \* q=NEWNODE(yunsuan);

match(ctoken.token\_texttype);

q->child=t;

(q->child)->rchilde=term(k);

t=q;

}

return t;

}

TREENODE \* term(TREENODE \* k)

{

TREENODE \* t=factor(k);

k=NULL;

while(ctoken.token\_texttype==TIMES||ctoken.token\_texttype==DIVISION)

{

TREENODE \* q=NEWNODE(yunsuan);

match(ctoken.token\_texttype);

q->child=t;

(q->child)->rchilde=factor(k);

t=q;

}

return t;

}

TREENODE \* factor(TREENODE \* k)

{

TREENODE \* t=NULL;

if(k!=NULL) //k为传递而来的variable;id k的NODEstring是变量

{

if(ctoken.token\_texttype==LP)//call

{

t=call(k); //如果当前的ctoken里面的string是括号，那么判定为函数调用

}

else//variable，等于本身

{

t=k; //不是函数调用

}

}

else

{

if(ctoken.token\_texttype==LP)

{

match(LP);

t=expression(); //得到一个变量id

match(RP); //匹配到右括号

}

else if(ctoken.token\_texttype==ID)//还是有可能是variable或call

{

t=variabler();//variable

if(ctoken.token\_texttype==LP)

{

t=call(k);

}

}

else if(ctoken.token\_texttype == INT\_CONST)

{

t=NEWNODE(INT\_CONSTS);

match(INT\_CONST);

}

else if(ctoken.token\_texttype == LONG\_CONST)

{

t=NEWNODE(LONG\_CONSTS);

match(LONG\_CONST);

}

else

{

printf("factor error\n");

programerror();

gettoken\_text(); //得到下一个节点数组tokenlist里面的结点

}

}

return t;

}

TREENODE \* call(TREENODE \* k)//函数调用

{

TREENODE \* t=NEWNODE(hanshudiaoyong); //函数调用元素k，从上面下降递归得到的

if(k!=NULL)

{

t->child=k;

}

match(LP);

if(ctoken.token\_texttype==RP)

{

match(RP);

return t;

}

else if(k!=NULL)

{

(t->child)->rchilde=function\_listformat();

match(RP);

}

return t;

}

TREENODE \* function\_listformat() //函数参数列表格式

{

TREENODE \* t=NEWNODE(hanshudiaoyongcanlist);

TREENODE \* p=NULL;

TREENODE \* q=NULL;

if(ctoken.token\_texttype!=RP)

{

p=expression();

q=p;

while(ctoken.token\_texttype==COMMA)

{

TREENODE \* s;

match(COMMA);

s=expression();

if(q!=NULL)

{

if(p==NULL)

{

p=s;

q=s;

}

else

{

q->rchilde=s;

q=s;

}

}

}

}

if(p!=NULL)

{

t->child=p;

}

return t;

}

void printspace(int n)//打印空格

{

int i;

for(i=0;i<n;i++)

{

printf(" ");

}

}

void printTREE(TREENODE \* t)//打印语法分析树

{

while(t != NULL)

{

printspace(blank); //打印行前空格

if(t->NODEkind == hong)

{

printf("宏定义\n");

}

else if(t->NODEkind == hong1)

{

printf("#include<stdio.h>\n");

}

else if(t->NODEkind == hong2)

{

printf("#include<stdlib.h\n");

}

else if(t->NODEkind == hong3)

{

printf("#include<string.h>\n");

}

else if(t->NODEkind == hong4)

{

printf("#include<file.h>\n");

}

else if(t->NODEkind == hong5)

{

printf("#include<math.h>\n");

}

else if(t->NODEkind == ints)

{

printf("int\n");

}

else if(t->NODEkind == longs)

{

printf("long\n");

}

else if(t->NODEkind == chars)

{

printf("char\n");

}

else if(t->NODEkind == floats)

{

printf("float\n");

}

else if(t->NODEkind==voids)

{

printf("void\n");

}

else if(t->NODEkind==ids)

{

printf("ID:%s\n",t->NODEstring);

}

else if(t->NODEkind==INT\_CONSTS)

{

printf("整型常数:%s\n",t->NODEstring);

}

else if(t->NODEkind==LONG\_CONSTS)

{

printf("长整型常数:%s\n",t->NODEstring);

}

else if(t->NODEkind==variable)

{

printf("赋值\n");

}

else if(t->NODEkind==shuzu)

{

printf("数组声明\n");

}

else if(t->NODEkind==hanshu)

{

printf("函数声明\n");

}

else if(t->NODEkind==hancanlist)

{

printf("函数参数列表\n");

}

else if(t->NODEkind==hancan)

{

printf("函数参数\n");

}

else if(t->NODEkind==fuheyuju)

{

printf("复合语句\n");

}

else if(t->NODEkind==ifs)

{

printf("if\n");

}

else if(t->NODEkind==elses)

{

printf("else\n");

}

else if(t->NODEkind==whiles)

{

printf("while\n");

}

else if(t->NODEkind==returns)

{

printf("return\n");

}

else if(t->NODEkind==breaks)

{

printf("break\n");

}

else if(t->NODEkind==continues)

{

printf("continue\n");

}

else if(t->NODEkind==fuzhi)

{

printf("赋值\n");

}

else if(t->NODEkind==yunsuan)

{

printf("运算:%s\n",t->NODEstring);

}

else if(t->NODEkind==shuzuyuansu)

{

printf("数组元素\n");

}

else if(t->NODEkind==hanshudiaoyong)

{

printf("函数调用\n");

}

else if(t->NODEkind==hanshudiaoyongcanlist)

{

printf("函数调用参数列表\n");

}

else

{

printf("未知\n");

}

blank++;//子树空格多一个

if(t->child!=NULL)

{

printTREE(t->child);

}

blank--;//右孩子

t=t->rchilde;

}

}

int fff = 0;

int flag1 = 0;

int flag2;

int fbp = 0;

char snc[1000][1000];

void savefile(TREENODE \* T)//保存文件辅助函数

{

TREENODE \* t;

t = T;

int ll;

int i;

while(t != NULL)

{

if(t->NODEkind == hong)

{

goto end;

}

else if(t->NODEkind == hong1)

{

strcpy(snc[flag1++],"#include<stdio.h>");

strcpy(snc[flag1++],"#");

fff++;

}

else if(t->NODEkind == hong2)

{

strcpy(snc[flag1++],"#include<stdlib.h>");

strcpy(snc[flag1++],"#");

fff++;

}

else if(t->NODEkind == hong3)

{

strcpy(snc[flag1++],"#include<string.h>");

strcpy(snc[flag1++],"#");

fff++;

}

else if(t->NODEkind == hong4)

{

strcpy(snc[flag1++],"#include<file.h>");

strcpy(snc[flag1++],"#");

fff++;

}

else if(t->NODEkind == hong5)

{

strcpy(snc[flag1++],"#include<math.h>");

strcpy(snc[flag1++],"#");

fff++;

}

else if(t->NODEkind == ints)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"int ");

fff++;

}

else if(t->NODEkind == longs)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"long ");

fff++;

}

else if(t->NODEkind == chars)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"char ");

fff++;

}

else if(t->NODEkind == floats)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"float ");

fff++;

}

else if(t->NODEkind==variable)

{

goto end;

}

else if(t->NODEkind==shuzu)

{

goto end;

}

else if(t->NODEkind==hanshu)

{

goto end;

}

else if(t->NODEkind==hancanlist)

{

goto end;

}

else if(t->NODEkind==hancan)

{

goto end;

}

else if(t->NODEkind==fuheyuju)

{

fbp++;

goto end;

}

else if(t->NODEkind==voids)

{

strcpy(snc[flag1++],"void ");

ll = fff+1;

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

if(!strcmp(tokenlist[ll].token\_textstring ,")"))

{

strcpy(snc[flag1++],")");

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

ll++;

ll++;

fff = ll;

}

else

goto end;

}

else if(t->NODEkind==ids)

{

if(!strcmp(tokenlist[fff-1].token\_textstring ,";"))

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

}

if(!strcmp(tokenlist[fff-1].token\_textstring ,"{"))

{

for(i=0;i<fbp;i++)

{;

strcpy(snc[flag1++]," ");

}

}

if(!strcmp(tokenlist[fff-1].token\_textstring ,")"))

{

for(i=0;i<fbp;i++)

{

strcpy(snc[flag1++]," ");

}

}

if(!strcmp(tokenlist[fff-1].token\_textstring ,"else"))

{

for(i=0;i<fbp+1;i++)

{

strcpy(snc[flag1++]," ");

}

}

strcpy(snc[flag1++],t->NODEstring);

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

while(!strcmp(tokenlist[fff].token\_textstring ,"}"))

{

for(i=0;i<fbp-1;i++);

strcpy(snc[flag1++]," ");;

strcpy(snc[flag1++],"}");

strcpy(snc[flag1++],"#");

fff++;

fbp--;

}

}

else if(!strcmp(tokenlist[fff].token\_textstring ,","))

{

strcpy(snc[flag1++],",");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"["))

{

strcpy(snc[flag1++],"[");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"("))

{

strcpy(snc[flag1++],"(");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"="))

{

strcpy(snc[flag1++],"=");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"=="))

{

strcpy(snc[flag1++],"==");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"+"))

{

strcpy(snc[flag1++],"+");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"-"))

{

strcpy(snc[flag1++],"-");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"\*"))

{

strcpy(snc[flag1++],"\*");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"/"))

{

strcpy(snc[flag1++],"/");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,">"))

{

strcpy(snc[flag1++],">");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"<"))

{

strcpy(snc[flag1++],"<");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,">="))

{

strcpy(snc[flag1++],">=");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"<="))

{

strcpy(snc[flag1++],"<=");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,")"))

{

strcpy(snc[flag1++],")");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"{"))

{

fbp++;

strcpy(snc[flag1++],"#");

for(i=0;i<fbp-1;i++)

strcpy(snc[flag1++]," ");

strcpy(snc[flag1++],"{");

strcpy(snc[flag1++],"#");

fff++;

}

}

}

else if(t->NODEkind==INT\_CONSTS)

{

strcpy(snc[flag1++],t->NODEstring);

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,"+"))

{

strcpy(snc[flag1++],"+");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"-"))

{

strcpy(snc[flag1++],"-");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"\*"))

{

strcpy(snc[flag1++],"\*");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"/"))

{

strcpy(snc[flag1++],"/");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"]"))

{

strcpy(snc[flag1++],"]");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

while(!strcmp(tokenlist[fff].token\_textstring ,"}"))

{

strcpy(snc[flag1++],"}");

strcpy(snc[flag1++],"#");

fff++;

fbp--;

}

}

}

}

else if(t->NODEkind==LONG\_CONSTS)

{

strcpy(snc[flag1++],t->NODEstring);

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,"+"))

{

strcpy(snc[flag1++],"+");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"-"))

{

strcpy(snc[flag1++],"-");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"\*"))

{

strcpy(snc[flag1++],"\*");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,"/"))

{

strcpy(snc[flag1++],"/");

fff++;

}

else if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

}

}

else if(t->NODEkind==ifs)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"if(");

fff++;

fff++;

}

else if(t->NODEkind==elses)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"else");

strcpy(snc[flag1++],"#");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,"{"))

{

strcpy(snc[flag1++],"{");

strcpy(snc[flag1++],"#");

fff++;

fbp++;

}

}

else if(t->NODEkind==whiles)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"while");

strcpy(snc[flag1++],"#");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,"("))

{

strcpy(snc[flag1++],"(");

fff++;

}

}

else if(t->NODEkind==returns)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"return ");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

while(!strcmp(tokenlist[fff].token\_textstring ,"}"))

{

fff++;

fbp--;

strcpy(snc[flag1++],"}");

strcpy(snc[flag1++],"#");

}

}

}

else if(t->NODEkind==breaks)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"break");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

while(!strcmp(tokenlist[fff].token\_textstring ,"}"))

{

strcpy(snc[flag1++],"}");

strcpy(snc[flag1++],"#");

fff++;

fbp--;

}

}

}

else if(t->NODEkind==continues)

{

for(i=0;i<fbp-1;i++)

{

strcpy(snc[flag1++]," ");

}

strcpy(snc[flag1++],"continue");

fff++;

if(!strcmp(tokenlist[fff].token\_textstring ,";"))

{

strcpy(snc[flag1++],";");

strcpy(snc[flag1++],"#");

fff++;

while(!strcmp(tokenlist[fff].token\_textstring ,"}"))

{

strcpy(snc[flag1++],"}");

strcpy(snc[flag1++],"#");

fff++;

fbp--;

}

}

}

else if(t->NODEkind==fuzhi)

{

goto end;

}

else if(t->NODEkind==yunsuan)

{

goto end;

}

else if(t->NODEkind==shuzuyuansu)

{

goto end;

}

else if(t->NODEkind==hanshudiaoyong)

{

goto end;

}

else if(t->NODEkind==hanshudiaoyongcanlist)

{

goto end;

}

else

{

goto end;

}

end:

if(t->child!=NULL)

{

savefile(t->child);

}

t=t->rchilde;

}

}

void save(FILE \* FP)//保存文件

{

int flag3;

flag3 = flag1-1;

int i;

for(i=0;i<flag3;i++)

{

if(!strcmp(snc[i],"#"))

fputc('\n',FP);

else

fprintf(FP,"%s",snc[i]);

}

}

### 附录2：

|  |  |
| --- | --- |
| 操作指令 | 功能 |
| 输入代码，回车以ctrl+z结尾 | 读入数据 |
| 若使用代码1，输入1 | 打印出语法分析树 |
| 若使用代码2，输入2，输入文件名A.txt | 以一定的格式保存文件 |
| 退出，输入0 | 退出系统 |

代码1：几乎包含了所有的词法分析和语法分析的需要，但是由于判断过多，我在保存文件时并没有加上那么多的判断，所以只适合打印语法分析树，代码如下

#include<stdio.h>

//int m;

int n;

//long c;

int fun(int a,int b)

{

int d;

c=123L;

d=0x123;

while(m == n)

{

if (a>b)

{n=a;

return n ;

}

else

{

m=(c+d)\*(b+a);

//continue;

return m;

}

}

}

float x;

/\*aa\*/

void main(void)

{

int a; /\*a\*/

a = fun(c,d); //aa

return 0;

}

代码2：代码段简单，和实验要求所给的代码段基本无异，可以打印语法分析树和保存文件，代码如下

#include<stdio.h>

int n;

int fun(int a,int b)

{

int m;

//int n;

if (a>b)

m=a;

else

m=b+a;

return m;

}

float x;