

**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： U201814604**

**姓 名： 黄俊淇**

**指导教师： 袁凌**

**报告日期： 2020.3.29**

**计算机科学与技术学院**

任务书

1. **设计内容**

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)等诸多领域有广泛的应用. 本设计需要采用形式化的方式，使用巴克斯（BNF）范式定义高级语言的词法规则（字符组成单词的规则）、语法规则（单词组成语句、程序等的规则）。再利用形式语言自动机的的原理，对源程序的文件进行词法分析，识别出所有单词；使用编译技术中的递归下降语法分析法，分析源程序的语法结构，并生成抽象语法树,最后可由抽象语法树生成格式化的源程序。

1. **设计要求**

1. 语言定义

选定C语言的一个子集，要求包含：

（1）基本数据类型的变量、常量，以及数组。不包含指针、结构，枚举等。

（2) 双目算术运算符（+-\*/%），关系运算符、逻辑与（&&）、逻辑或（||）、赋值运算符。不包含逗号运算符、位运算符、各种单目运算符等等。

（3）函数定义、声明与调用。

（4）表达式语句、复合语句、if语句的2种形式、while语句、for语句，return语句、break语句、continue语句、外部变量说明语句、局部变量说明语句。

（5）编译预处理（宏定义，文件包含）

（6）注释（块注释与行注释）

2. 单词识别

设计DFA的状态转换图（参见实验指导），实验时给出DFA，并解释如何在状态迁移中完成单词识别（每个单词都有一个种类编号和单词的字符串这2个特征值），最终生成单词识别（词法分析）子程序。

**注：含后缀常量，以类型不同作为划分标准种类编码值，例如123类型为int，123L类型为long，单词识别时，种类编码应该不同；但0x123和123类型都是int，种类编码应该相同。**

3. 语法结构分析

（1）外部变量的声明；

（2）函数声明与定义；

（3）局部变量的声明；

（4）语句及表达式；

（5）生成（1)-(4)（包含编译预处理和注释）的抽象语法树并显示。

4. 按缩进编排生成源程序文件。

目录

[任务书 2](#_Toc36410152)

[目录 4](#_Toc36410153)

[1、 引言 6](#_Toc36410154)

[1.1课题背景与意义 6](#_Toc36410155)

[1.2国内外研究现状 6](#_Toc36410156)

[1.3课程设计中的主要工作 6](#_Toc36410157)

[2、 系统需求与总体设计 8](#_Toc36410158)

[2.1系统需求分析 8](#_Toc36410159)

[2.2系统总体设计 8](#_Toc36410160)

[3、系统详细设计 10](#_Toc36410161)

[3.1有关数据结构的定义 10](#_Toc36410162)

[3.2主要算法设计 11](#_Toc36410163)

[4、系统实现与测试 16](#_Toc36410164)

[4.1系统实现 16](#_Toc36410165)

[4.2系统测试 17](#_Toc36410166)

[5、总结与展望 22](#_Toc36410167)

[5.1全文总结 22](#_Toc36410168)

[5.2工作展望 22](#_Toc36410169)

[6、体会 23](#_Toc36410170)

[参考文献 24](#_Toc36410171)

[附录一 源代码 25](#_Toc36410172)

[附录二 程序使用说明 49](#_Toc36410173)

# 引言

## 1.1课题背景与意义

#### 1.1.1课题背景

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处理工具,比如源程序浏览器、智能编辑器、语言翻译器等

#### 1.1.2课题意义

本课题通过对一段代码进行词法分析，语法分析形成抽象语法树，来更好地理解高级语言的语法特征。

## 1.2国内外研究现状

现在有很多AST工具，例如ANTLR，它是有Sanfraneiseo大学TerenceParr等人开发的一种分析器自动生成工具，它同时集成了词法分析器生成、语法分析器生成、抽象语法树生成和抽象语法树遍历框架生成功能，并且能够支持多种高级语言。

## 1.3课程设计中的主要工作

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。

1. 词法分析的过程，就是在读取源程序的文本文件的过程中，识别出一个个的单词。
2. 语法分析的过程，采用的实现方法是编译技术中的递归下降子程序法，递归下降子程序法是一种非常简洁的语法结构分析算法，基本上是每个语法成分对应一个子程序，每次根据识别出的前几个单词，明确对应的语法成分，调用相应子程序进行语法结构分析。

# 系统需求与总体设计

## 2.1系统需求分析

用户需要检测一段代码的单词是否正确，并看到这段代码的抽象语法树。

## 2.2系统总体设计

1、词法分析如图2.2.1所示

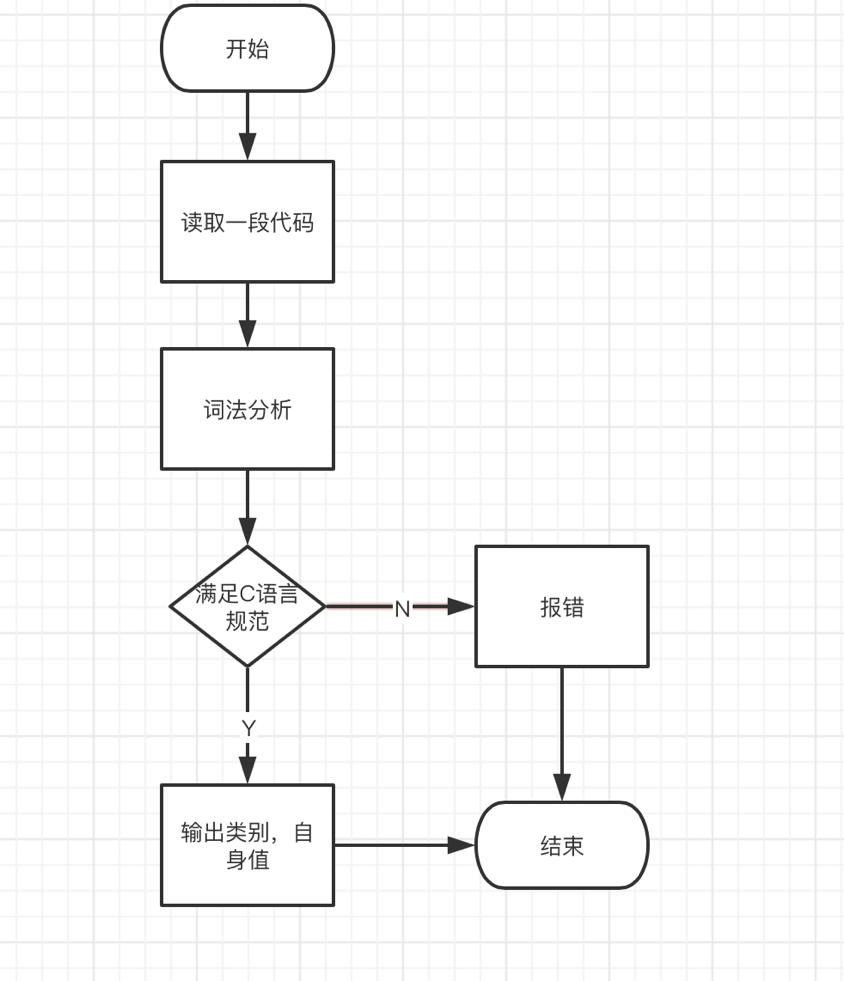


图2.2.1词法分析

2、语法分析如图2.2.2所示

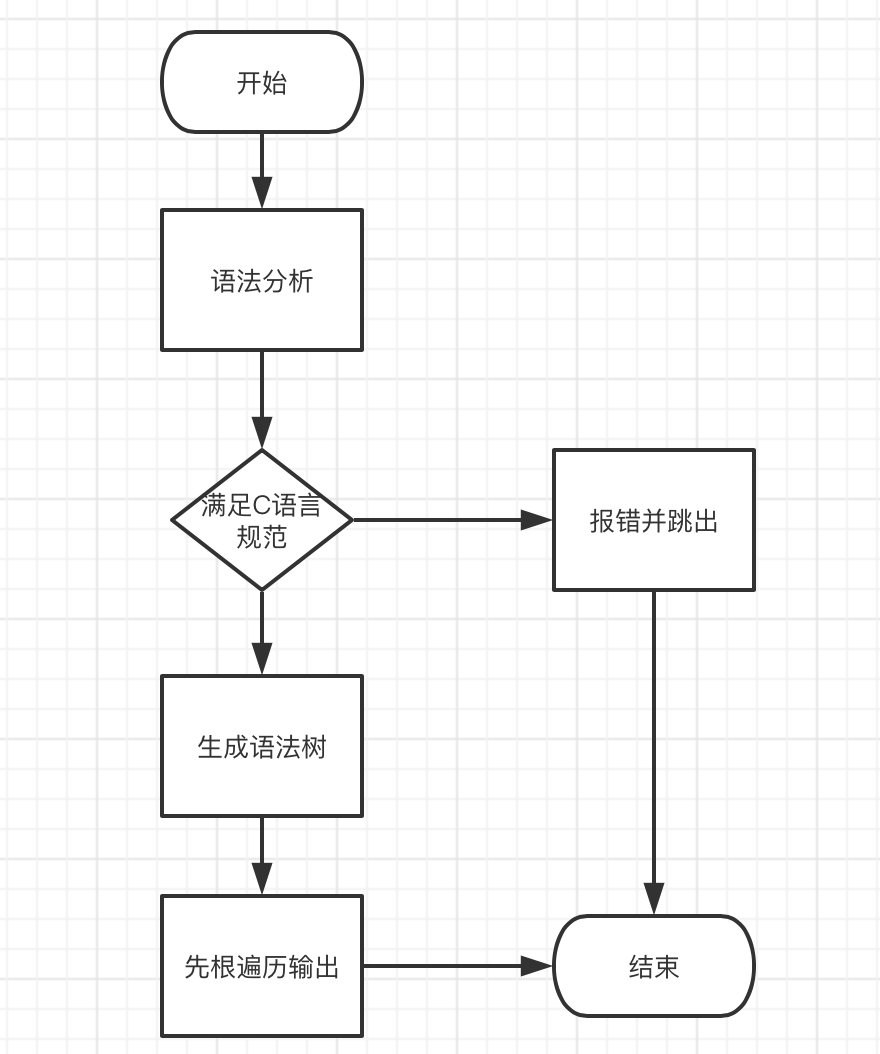


图2.2.2语法分析

# 3、系统详细设计

## 3.1有关数据结构的定义

1、本系统要处理的数据为一段用C语言写的代码，包括外部变量定义，内部变量定义，函数声明，函数类型，函数体，复合语句以及表达式。

（1）变量定义包括变量的类型和变量名称，通过本系统确认它是否满足C语言的规范，满足则输出。

（2）函数包括函数类型，函数声明，以及函数体，通过本系统可确认一条语句是函数声明还是函数定义。

（3）复合语句包括if-else语句，while语句，for语句，break语句，以及嵌套结构和循环结构，通过本系统可确认一段代码是否存在语法错误。

（4）表达式包括变量名以及运算符，通过本系统可确认一句表达式的运算先后顺序。

2、多种数据的关联如图3.1.1所示

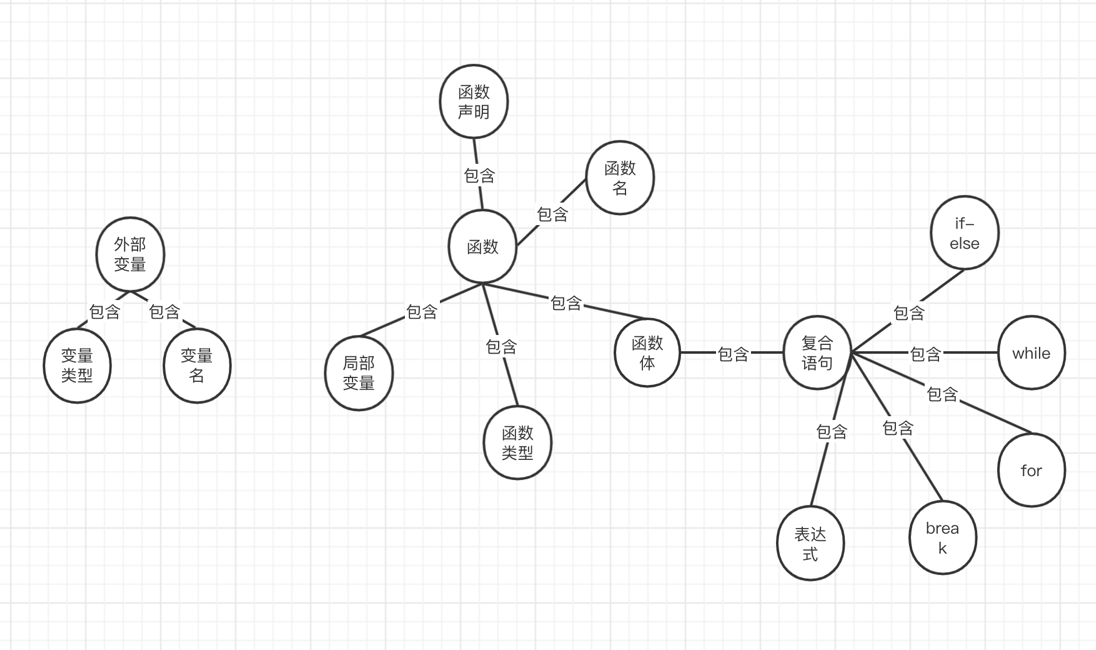


图3.1.1数据关联

## 3.2主要算法设计

1、词法分析

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。为唯一确定各单词的种类编码，可通过枚举类型定义各类单词的种类编号：

enum token\_kind{INT=0,CHAR,FLOAT,WHILE,IF,ELSE,BREAK,FOR,RETURN,CONTINUE,ERROR\_TOKEN,IDENT,INT\_CONST,EQ,PLUS,PLUSPLUS,MINUS,MINUSMINUS,MULTIPLY,DIVIDE,ASSIGN,SEMI,COMMA,MOD,AND,OR,LP,RP,BIG,SMALL,EM,LPP,RPP,EXP};

INT:整型；CHAR:字符型；FLOAT:浮点型；WHILE:循环语句；IF,ELSE:分支语句；BREAK:跳转语句；FOR:循环语句;RETURN:返回语句；CONTINUE:循环语句；ERROR\_TOKEN:报错；IDENT:标识符；INT\_CONST：整型常数；EQ：==；PLUS：加法；PLUSPLUS：自增；MINUS：减法；MINUSMINUS：自减；MULTIPLY：乘法；DIVIDE：除法；ASSIGN：赋值；SEMI：分号；COMMA：逗号；MOD：求余；AND：逻辑与；OR：逻辑非；LP：左括号；RP：右括号；BIG：大于号；SMALL：小于号；EM：！=；LPP：左大括号；RPP：右大括号；EXP：表达式开始符号。

另外对有些单词，仅有种类编码是不够的，如标识符abc，词法分析返回IDENT，但标识符的字符串值abc需要保存在一个字符数组中token\_text，称为单词的自身值，token\_text是一个全局变量；同样123，词法分析返回INT\_CONST，数字字符串123要保存在token\_text中，以备后续处理。

流程图如图3.2.1

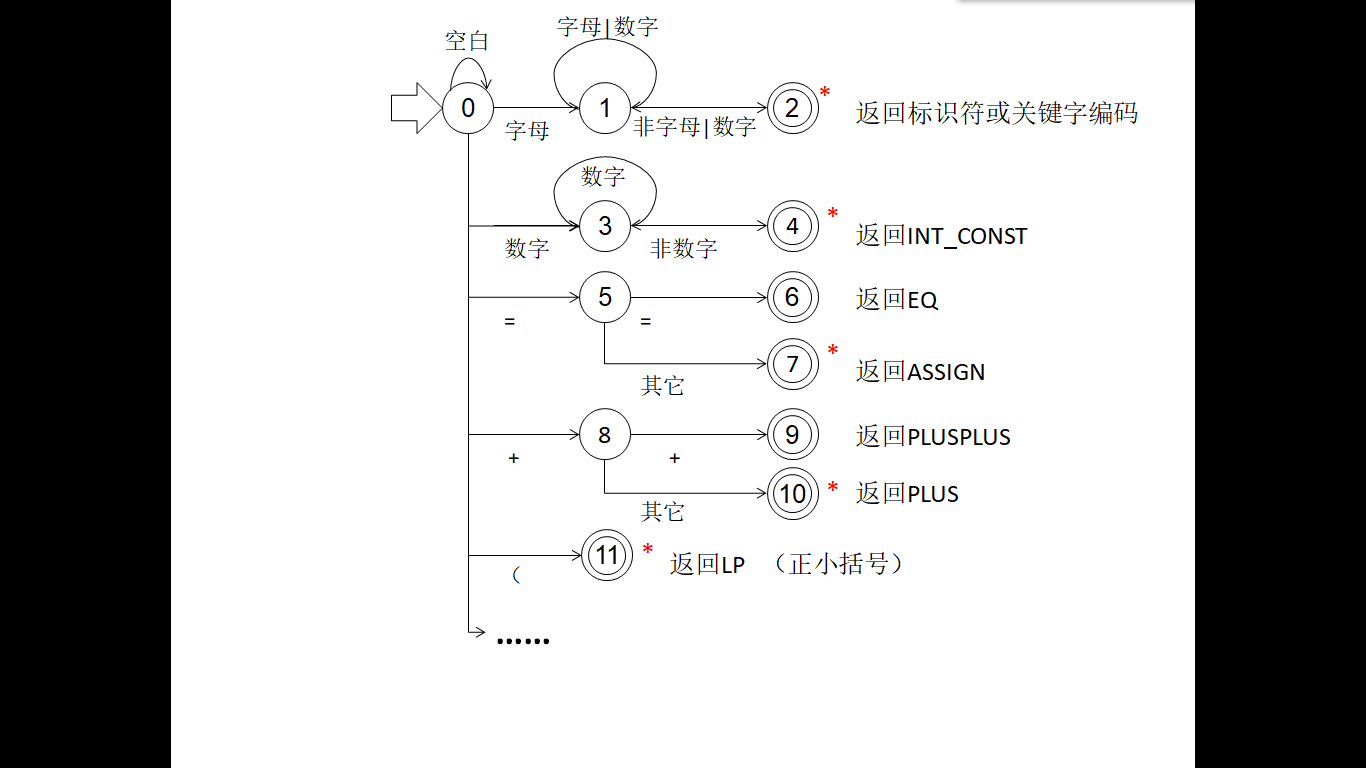


图3.2.1词法分析流程图

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，单词自身值保存在全局变量token\_text中。结束状态上标有星号的，表示从源程序文件中多读取了一个字符，这个字符可能是下一个单词的一部分，需要退回到文件的输入缓冲区中。根据这个状态转换图，可以设计出如下词法分析的函数。每调用一次得到一个单词的种类码和自身值。

2、语法分析

抽象语法树统一使用三叉树结构

1. 变量定义语法树，包括外部变量和内部变量，如图3.2.2

……

i，j的定义i部分

……

x，y的定义i部分

……

函数定义部分

图3.2.2变量定义语法树

局部变量类似。

（2）复合语句语法树如图3.2.3

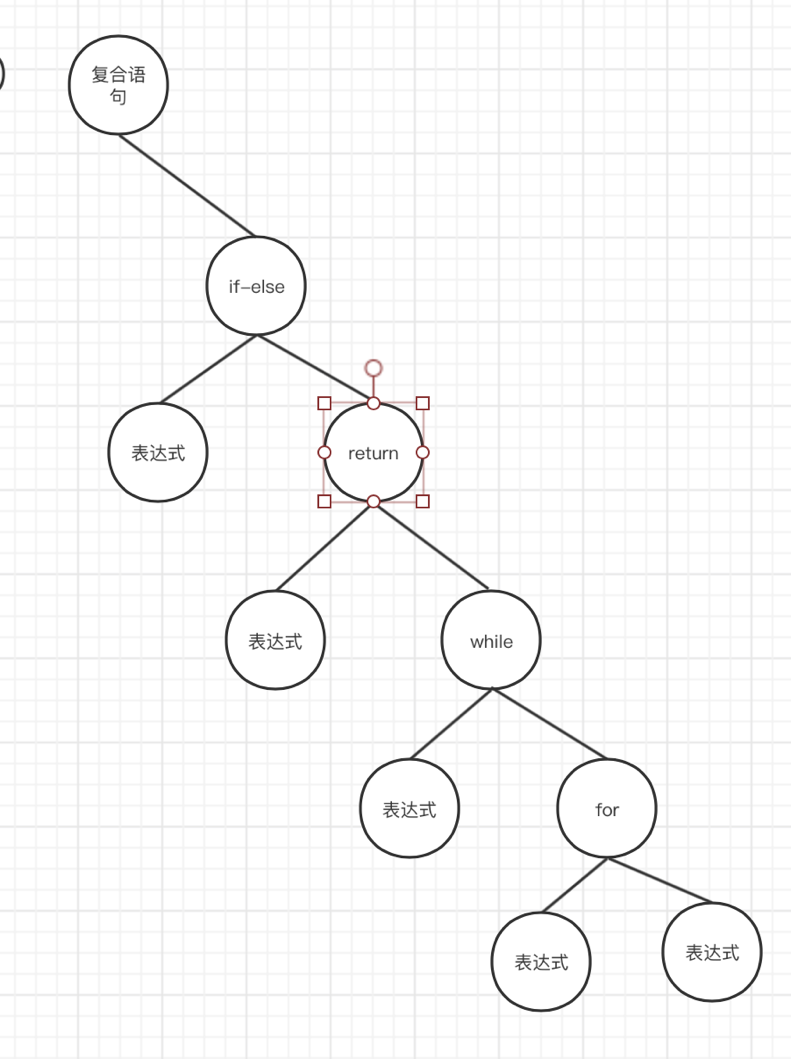


图3.2.3复合语句抽象语法树

（3）函数抽象语法树如图3.2.4

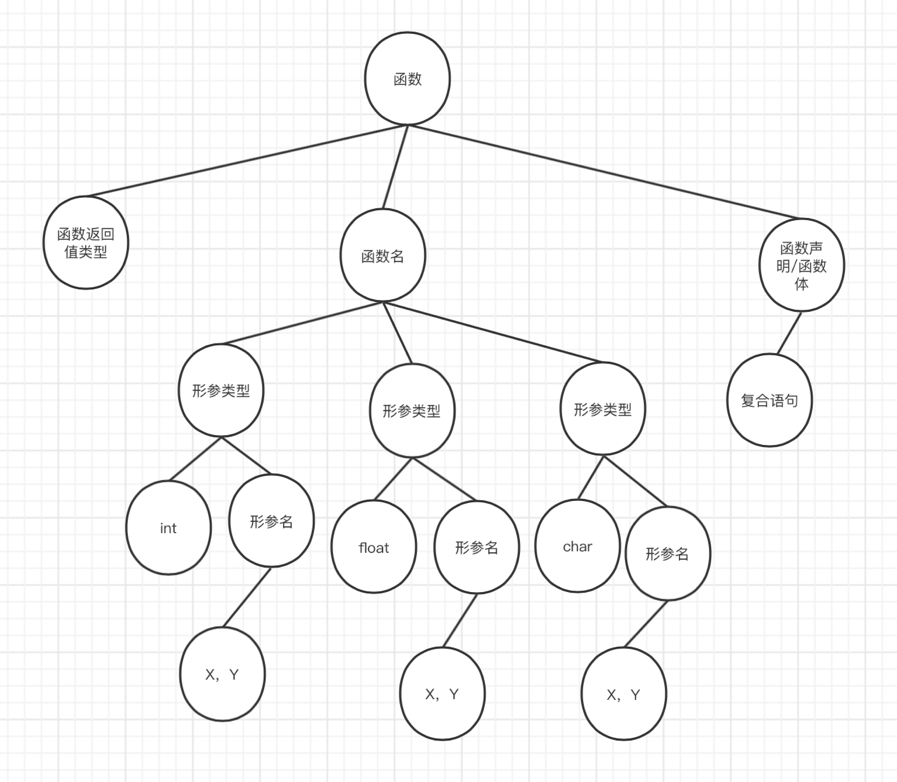


图3.2.4函数抽象语法树

（4）表达式抽象语法树

表达式符号优先级表如下图所示

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+** | **-** | **\*** | **/** | **（** | **）** | **=赋值** | **大小于** | **==和！=** | **#** |
| **+** | **>** | **>** | **<** | **<** | **<** | **>** |  | **>** | **>** | **>** |
| **-** | **>** | **>** | **<** | **<** | **<** | **>** |  | **>** | **>** | **>** |
| **\*** | **>** | **>** | **>** | **>** | **<** | **>** |  | **>** | **>** | **>** |
| **/** | **>** | **>** | **>** | **>** | **<** | **>** |  | **>** | **>** | **>** |
| **（** | **<** | **<** | **<** | **<** | **<** | **=** |  | **>** | **>** | **>** |
| **）** | **>** | **>** | **>** | **>** | **>** |  |  | **>** | **>** | **>** |
| **=赋值** | **<** | **<** | **<** | **<** | **<** |  | **<** | **<** | **<** | **>** |
| **大小于** | **<** | **<** | **<** | **<** | **<** | **>** |  | **>** | **>** | **>** |
| **==和！=** | **<** | **<** | **<** | **<** | **<** | **>** |  | **<** | **>** | **>** |
| **#** | **<** | **<** | **<** | **<** | **<** |  | **<** | **<** | **<** | **=** |

表1优先级表

根据优先级表创建表达式抽象语法树。

例如表达式语句a+b\*10；其表达式部分的抽象语法树如图3.2.5所示。

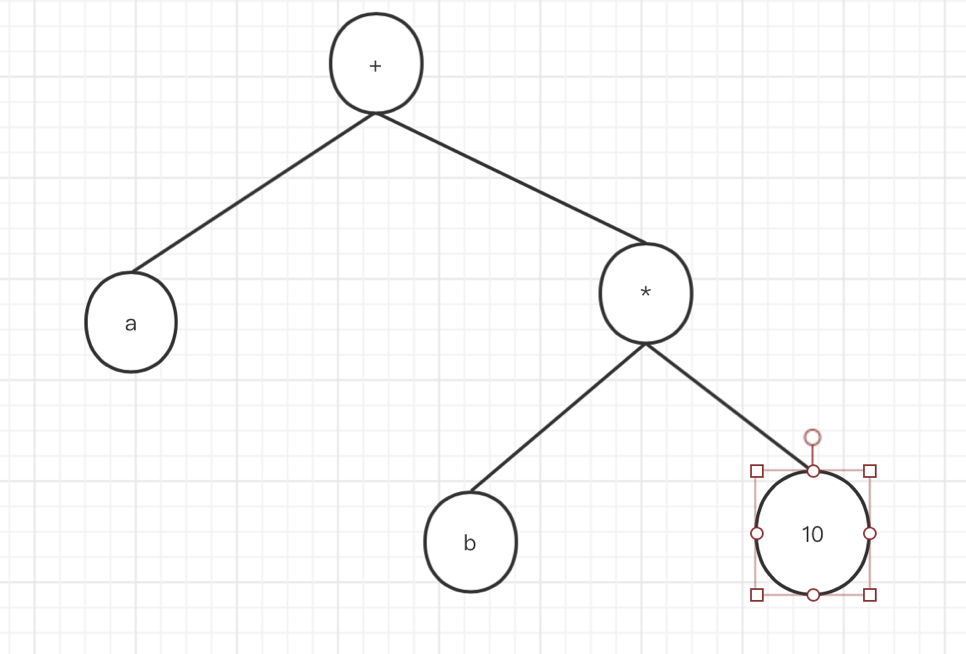


图3.2.5表达式抽象语法树

# 4、系统实现与测试

## 4.1系统实现

1、软硬件环境

Xcode 10.3

Macbook pro2018

2、数据结构

char keywords[10][10]={"int","char","float","while","if","else","break","for","return","continue"};

enum token\_kind{INT=0,CHAR,FLOAT,WHILE,IF,ELSE,BREAK,FOR,RETURN,CONTINUE,ERROR\_TOKEN,IDENT,INT\_CONST,EQ,PLUS,PLUSPLUS,MINUS,MINUSMINUS,MULTIPLY,DIVIDE,ASSIGN,SEMI,COMMA,MOD,AND,OR,LP,RP,BIG,SMALL,EM,LPP,RPP,EXP};

char token\_text[100];

typedef struct ASTnode{

char data[100];

struct ASTnode \*lchild;

struct ASTnode \*rchild;

struct ASTnode \*midchild;

}ASTnode,\*ASTTree;

typedef struct stack{

int \*base;

int \*top;

}stack;

3、函数声明

ASTTree funcDef(FILE \*fp)；//处理代码中的函数体的函数

char MYprecede(char \*lastc,char \*token\_text)；//优先级函数

ASTTree MYexp(FILE \*fp)；//表达式求值函数

ASTTree complex(FILE\*fp)；//处理复合语句函数

ASTTree ExtVarDef(FILE \*fp)；//处理外部变量函数

ASTTree ExtVarDef1(FILE \*fp)；//处理内部变量函数

int Pre\_order(ASTTree T, int (\*Visit)(ASTTree T))//先根遍历函数

详细程序见附录

## 4.2系统测试

1、测试代码

int a,b;

int fun(int a;float b){

a=123;

if(a>b) m=a;

else m=b;

while(a>b);

for(i=0;i<9;i++);

int a,b;

# a+b\*10

return m;

}

2、测试大纲

（1）词法分析

|  |  |  |
| --- | --- | --- |
| 测试状态 | 测试数据 | 预测结果 |
| 正常测试 | 所有代码均满足C语言规范 | 输出：单词类别 单词值 |
| 特殊代码测试 | 输入16进制数 | 输出：整型常量 单词值 |
| 错误样例 | 输入数字开头的标识符 | 输出：错误 |

1. 语法分析

|  |  |  |
| --- | --- | --- |
| 测试状态 | 测试数据 | 预测结果 |
| 无嵌套结构测试 | 所有循环语句中不含有其他循环语句 | 输出：抽象语法树，循环语句不含循环体 |
| 含嵌套结构测试 | While循环中加入if-else语句，while语句，for语句 | 输出：抽象语法树，循环体内也生成语法树 |
| 函数声明测试 | 输入以分号结尾的函数语句 | 输出：函数声明 |
| 错误样例 | 输入以句号结尾的函数语句 | 输出：错误 |

3、测试结果记录

（1）词法分析如图4.2.1a,b,c,d所示

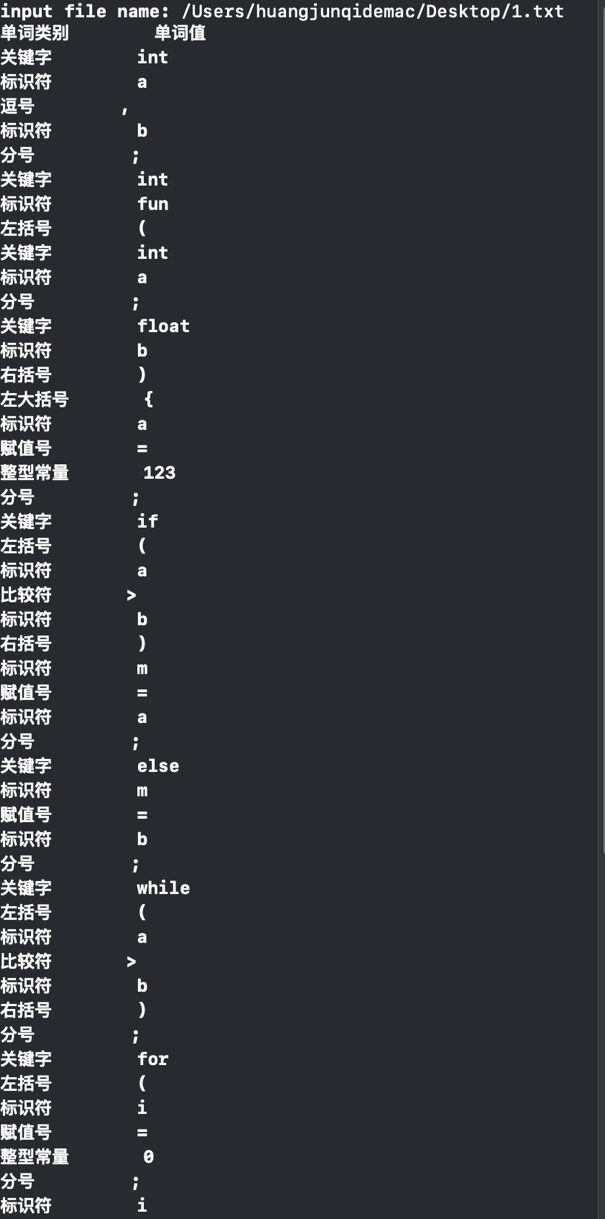


图4.2.1a 词法分析

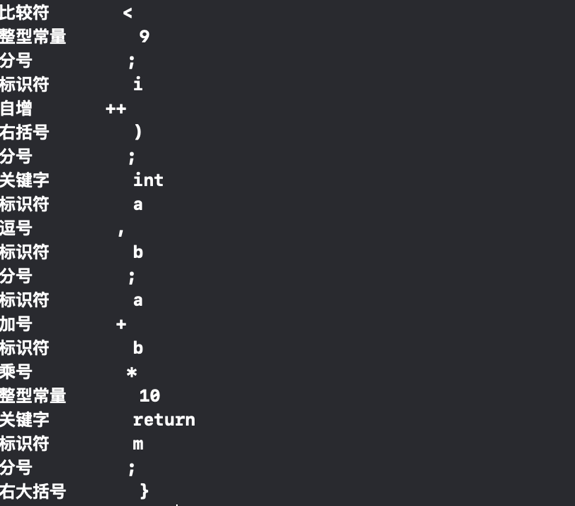


图4.2.1b 词法分析

对于整型常量，有不同输入方法，如123，0x123都为整型常量，故将源代码中的“a=123;”替换为”a=0x123”再进行测试，结果如图4.2.1c所示（只截取部分内容）

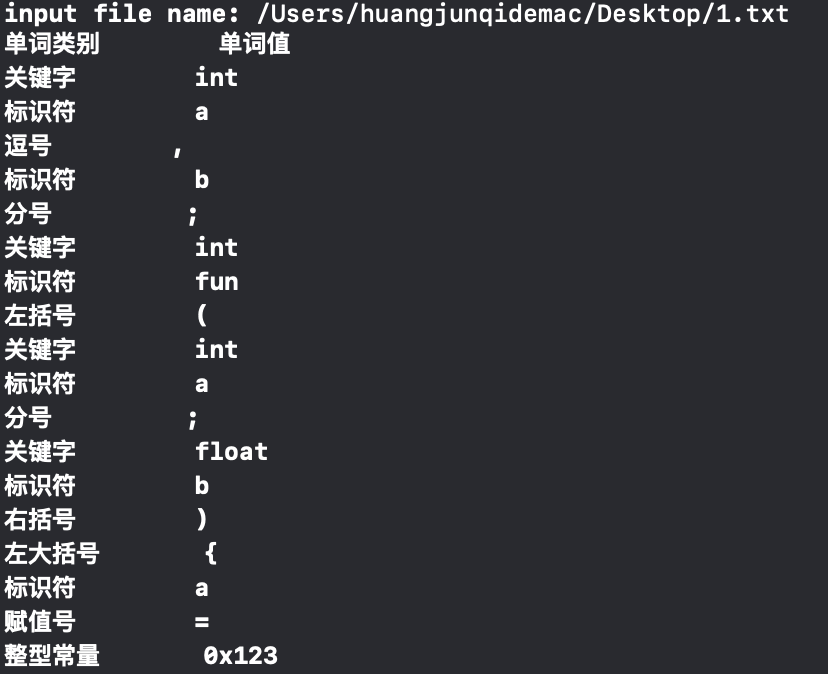


图4.2.1c 不同输入方法同一类别的词法分析

C语言中，标识符第一个字符不能是数字，故将源代码中的外部变量定义“int a；”改为“int 1a；”再进行测试，测试结果如图4.2.1d所示

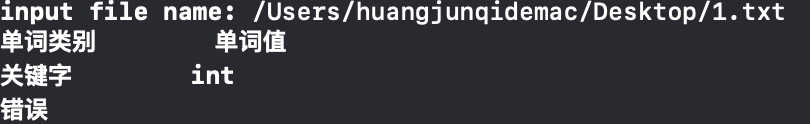


图4.2.1d 标识符错误

（2）语法分析如图4.2.2所示

无嵌套结构测试如图4.2.2a所示



接上图

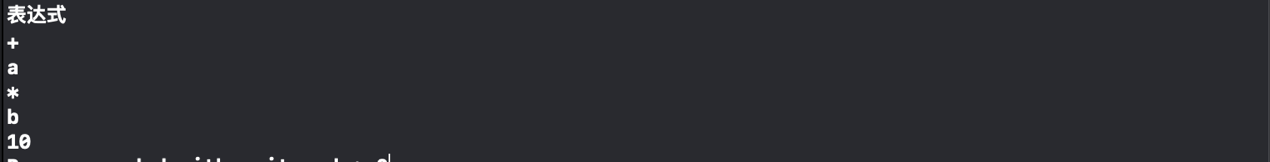


图4.2.2a 无嵌套结构

含嵌套结构测试如图4.2.2b所示 将源代码的while语句扩展为：

while(a>b){

if(a>b) x=a;

else x=b;

while(c>d);

for(j=0;j<9;j++);

# a\*b+10

return x;

}

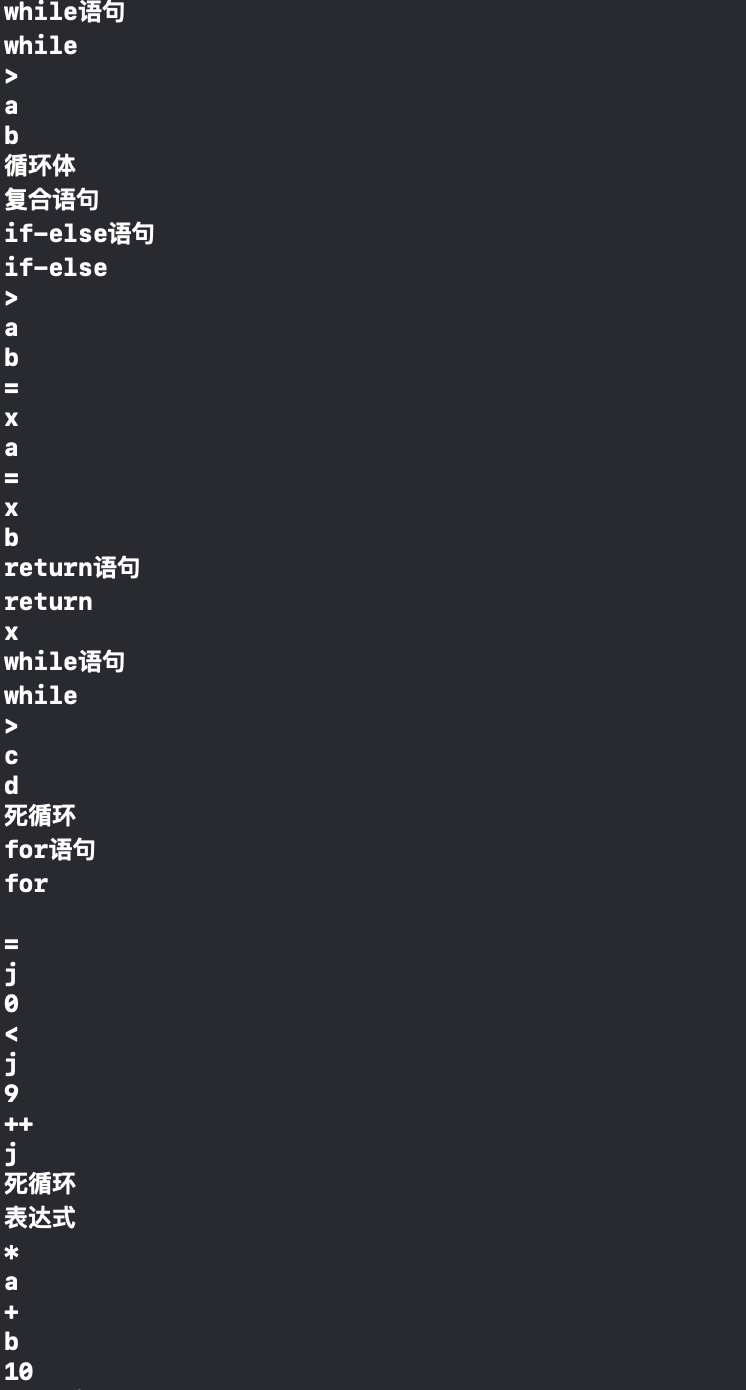


图4.2.2b 含嵌套结构测试

函数声明测试如图4.2.2c所示，代码如下

int a,b;

int fun(int a;float b);

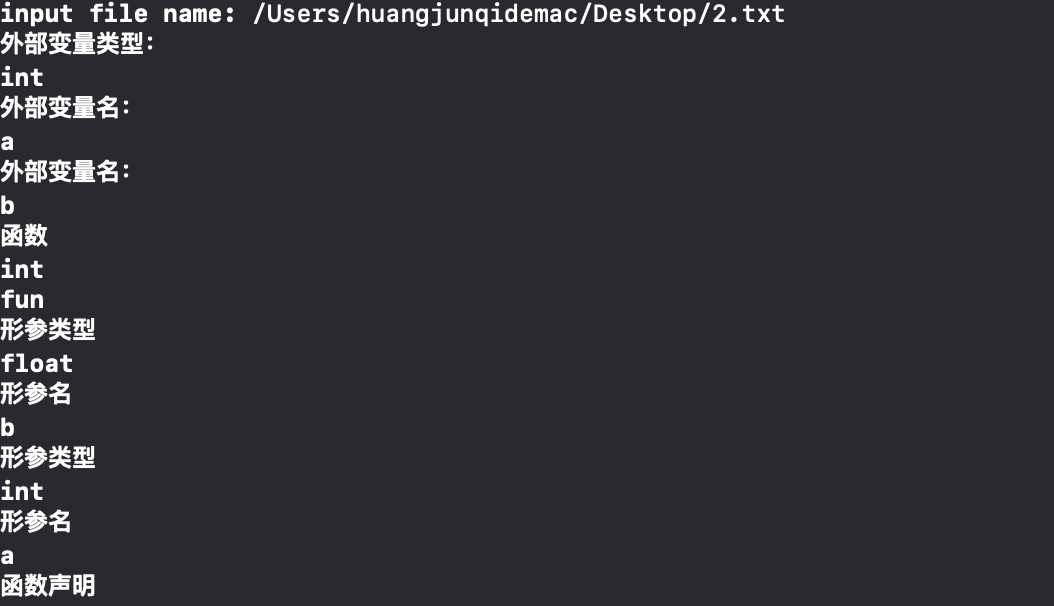


图4.2.2c 函数声明测试

含报错语句测试如图4.2.2d所示

测试代码如下：

int a,b;

int fun(int a;float b).//句号

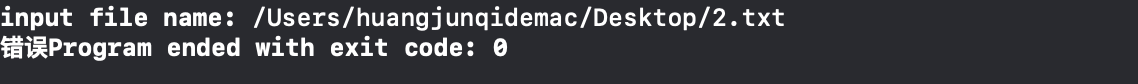


图4.2.2d 含报错语句测试

（4）运行后，可以看到所有输出结果皆满足测试大纲中的预测结果。

# 5、总结与展望

## 5.1全文总结

1. 对于用户输入的代码进行分析，满足C语言规范的单词可以输出其类别和自身值。
2. 对于错误单词可以在相应位置报错。
3. 语法分析基本满足任务书所需功能，如嵌套结构，函数声明与函数体的区分，先根遍历输出抽象语法树等。
4. 对于用户输入的格式有较大要求，无法像C语言编译器可以有较大自由度的输入格式。
5. 交互界面比较简洁，输出结果不够美观。
6. 所有抽象语法树的数据结构为三叉树，对于复合语句，只能生成规定关键字的语法树：while，for，return，if-else，break。除此之外的关键字的复合语句，本系统无法生成其对应的抽象语法树。
7. 在语法分析时，如果出现语法错误，无法在相应位置报错。
8. 由于编程环境问题，采用文本编辑器存放测试用例。

## 5.2工作展望

在今后的研究中，将围绕以下几个方面工作：

1. 增加新的关键字的抽象语法树。
2. 使交互界面以及输出结果更加美观。
3. 可以在相应位置报错。
4. 如果出现语法错误，增加提示：用户可能需要的语句。

# 6、体会

这次课设综合运用了大一，大二学的C语言和数据结构，上手比较困难，指导书也比较晦涩。这个课题的功能也比较难实现，特别是复合语句的抽象语法树生成，如果用二叉树结构，代码非常复杂，还不容易达到预期效果。后期我更改了数据结构，采用三叉树结构，就很好的契合了任务书中要求实现的功能，这个大大减少了我的工作量，因为这次要求的变量类型就只有三个，int，char，float，因此只需要预先分配好这三个变量类型的空间，如果输入数据中有这三个变量类型，就可以直接分配到对应空间，如果没有，就可以释放预先分配好的空间。其次就是函数的抽象语法树，三叉树结构可以很好的存放：函数类型，函数名，函数体，当然，形参我是放在函数名下的三个子树。而且，三叉树的遍历也较为简单，与二叉树类似。

总的来说，这次课设前期进展不是很理想，但改变数据结构之后，课设的工作就进行的比较顺利了。而且这个系统还有许多待改进的功能，超出任务书的要求，这个留待以后再去优化吧。

# 参考文献

[1] 曹计昌，卢萍，李开. C语言与程序设计. 电子工业出版社，2013

[2] 严蔚敏等.数据结构（C语言版）.清华大学出版社，

[3] 程序设计综合课程设计指导-源程序格式处理

[4] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++. Second Edition,Calvin College,2005

附录一 源代码

源代码

main.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "预定义常量.h"

#include "词法分析.h"

#include "Tree\_function.h"

#include "variable.h"

#include "complex\_sentences.h"

#include "function.h"

#include "expression.h"

#include "栈.h"

int main() {

FILE \*fp;

char filename[30];

int num;

int w;

ASTTree root,T,T1;

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL)

{

printf("File open error\n ");

return -1;

}

printf("单词类别 单词值\n");

while(!feof(fp)){

num=gettoken(fp);

switch(num){

case INT:printf("关键字 %s\n",token\_text);

break;

case CHAR:printf("关键字 %s\n",token\_text);

break;

case FLOAT:printf("关键字 %s\n",token\_text);

break;

case WHILE:printf("关键字 %s\n",token\_text);

break;

case IF:printf("关键字 %s\n",token\_text);

break;

case ELSE:printf("关键字 %s\n",token\_text);

break;

case BREAK:printf("关键字 %s\n",token\_text);

break;

case FOR:printf("关键字 %s\n",token\_text);

break;

case RETURN:printf("关键字 %s\n",token\_text);

break;

case CONTINUE:printf("关键字 %s\n",token\_text);

break;

case INCLUDE:printf("关键字 %s\n",token\_text);

break;

case hfile:printf("头文件 %s\n",token\_text);

break;

case ERROR\_TOKEN:printf("错误\n");

break;

case IDENT:printf("标识符 %s\n",token\_text);

break;

case INT\_CONST:printf("整型常量 %s\n",token\_text);

break;

case EQ:printf("赋值号 %s\n",token\_text);

break;

case PLUS:printf("加号 +\n");

break;

case PLUSPLUS:printf("自增 ++\n");

break;

case MINUS:printf("减号 -\n");

break;

case MINUSMINUS:printf("自减 --\n");

break;

case MULTIPLY:printf("乘号 \*\n");

break;

case DIVIDE:printf("除号 /\n");

break;

case SEMI:printf("分号 ;\n");

break;

case COMMA:printf("逗号 ,\n");

break;

case MOD:printf("求余号 %s\n",token\_text);

break;

case AND:printf("逻辑与 &&\n");

break;

case OR:printf("逻辑或 ||\n");

break;

case LP:printf("左括号 (\n");

break;

case RP:printf("右括号 )\n");

break;

case ASSIGN:printf("赋值号 =\n");

break;

case SMALL:printf("比较符 <\n");

break;

case BIG:printf("比较符 >\n");

break;

case EM:printf("比较符 !=\n");

break;

case LPP:printf("左大括号 {\n");

break;

case RPP:printf("右大括号 }\n");

break;

case EXP:printf("井号 #\n");

break;

}

}

fclose(fp);

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL)

{

printf("File open error\n ");

return -1;

}

while(!feof(fp)){

T1=pre(fp);

if(T1==NULL) printf("预处理错误");

root=ExtVarDef(fp);

T=funcDef(fp);

//if(root==NULL) {printf("错误");break;}

if(T==NULL) {printf("错误");break;}

Pre\_order(T1,Visit);

Pre\_order(root,Visit);

Pre\_order(T,Visit);

w=gettoken(fp);

}

fclose(fp);

return 0;

}

栈.h

#ifndef \_\_h

#define \_\_h

void myinitstack(stack \*int\_sta){

int\_sta->base=(int \*)malloc(sizeof(int)\*100);

int\_sta->top=int\_sta->base;

}

void deletestack(stack \*int\_sta){

free(int\_sta->base);

}

void push(stack \*int\_sta,int i){

\*(int\_sta->top++)=i;

}

int pop(stack \*int\_sta,int \*i){

if(int\_sta->top==int\_sta->base) return 0;

else{

\*i=\*(--int\_sta->top);

return 1;

}

}

#endif /\* \_\_h \*/

#ifndef expression\_h

#define expression\_h

#include "预定义常量.h"

#include "栈.h"

char MYprecede(char \*lastc,char \*token\_text){

char precede[10][10]=

{ // + - \* / % （ ） = >和< ==和!=

/\* + \*/ {'=','>','<','<','<','<','>','?','>','>'},

/\* - \*/ {'>','=','<','<','<','<','>','?','>','>'},

/\* \* \*/ {'>','>','=','>','>','<','>','?','>','>'},

/\* / \*/ {'>','>','>','=','>','<','>','?','>','>'},

/\* % \*/ {'>','>','<','<','=','<','>','?','>','>'},

/\* ( \*/ {'<','<','<','<','<','<','=','?','<','>'},

/\* ) \*/ {'>','>','>','>','>','>','?','?','>','>'},

/\* = \*/ {'<','<','<','<','<','<','?','<','<','<'},

/\* >和< \*/ {'<','<','<','<','<','<','>','?','>','<'},

/\*!=和==\*/ {'<','<','<','<','<','<','<','<','<','<'}

};

int i=0,j=0;

if(strcmp(lastc,"+")==0) i=0;

if(strcmp(lastc,"-")==0) i=1;

if(strcmp(lastc,"\*")==0) i=2;

if(strcmp(lastc,"/")==0) i=3;

if(strcmp(lastc,"%")==0) i=4;

if(strcmp(lastc,"(")==0) i=5;

if(strcmp(lastc,")")==0) i=6;

if(strcmp(lastc,"=")==0) i=7;

if((strcmp(lastc,">")==0)||(strcmp(lastc,">")==0)) i=8;

if((strcmp(lastc,"!=")==0)||(strcmp(lastc,"==")==0)) i=9;

if(strcmp(token\_text,"+")==0) j=0;

if(strcmp(token\_text,"-")==0) j=1;

if(strcmp(token\_text,"\*")==0) j=2;

if(strcmp(token\_text,"/")==0) j=3;

if(strcmp(token\_text,"%")==0) j=4;

if(strcmp(token\_text,"(")==0) j=5;

if(strcmp(token\_text,")")==0) j=6;

if(strcmp(token\_text,"=")==0) j=7;

if(strcmp(token\_text,"!=")==0) j=8;

if((strcmp(token\_text,">")==0)||(strcmp(token\_text,">")==0)) i=8;

if((strcmp(token\_text,"!=")==0)||(strcmp(token\_text,"==")==0)) i=9;

return precede[i][j];

}

ASTTree MYexp(FILE \*fp){

int w;

char token\_text0[100];

char lastc[100];

ASTTree root = NULL,p=NULL;

//stack \*op = NULL;

//myinitstack(op);

(root)=(ASTTree)malloc(sizeof(ASTnode));

(root)->lchild=NULL;

(root)->rchild=NULL;

(root)->midchild=NULL;

w=gettoken(fp);

//printf("aaa%s",token\_text);

//if(w!=INT\_CONST||w!=IDENT) return NULL;

strcpy(token\_text0,token\_text);

w=gettoken(fp);

if(w==BIG) strcpy((root)->data,">");

if(w==SMALL) strcpy(root->data,"<");

if(w==EM) strcpy(root->data,"!=");

if(w==EQ) strcpy(root->data,"==");

if(w==ASSIGN) strcpy(root->data,"=");

if(w==PLUS) strcpy(root->data,"+");

if(w==MINUS) strcpy(root->data,"-");

if(w==MULTIPLY) strcpy(root->data,"\*");

if(w==DIVIDE) strcpy(root->data,"/");

if(w==PLUSPLUS) strcpy(root->data,"++");

if(w==MINUSMINUS) strcpy(root->data,"--");

if(w==SEMI||w==RP){

strcpy(root->data,token\_text0);

return root;

}

(root)->lchild=(ASTTree)malloc(sizeof(ASTnode));

(root)->lchild->lchild=NULL;

(root)->lchild->rchild=NULL;

(root)->lchild->midchild=NULL;

strcpy((root)->lchild->data,token\_text0);

strcpy(lastc,token\_text);

//if(w==MULTIPLY) printf("hjq");

w=gettoken(fp);

if(w==RP) { return root;}

strcpy(token\_text0,token\_text);//

w=gettoken(fp);

//if(w==MULTIPLY) printf("11%s",token\_text);

if(w==SEMI||w==RP){

(root)->rchild=(ASTTree)malloc(sizeof(ASTnode));

(root)->rchild->lchild=NULL;

(root)->rchild->rchild=NULL;

(root)->rchild->midchild=NULL;

strcpy((root)->rchild->data,token\_text0);}

else if(w>=EQ&&w<=SMALL){

//printf("%c",MYprecede(lastc,token\_text));

if(MYprecede(lastc,token\_text)=='<'){

p=(ASTTree)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->midchild=NULL;

p->rchild=NULL;

if(w==PLUS) strcpy(p->data,"+");

if(w==MINUS) strcpy(p->data,"-");

if(w==MULTIPLY) strcpy(p->data,"\*");

if(w==DIVIDE) strcpy(p->data,"/");

if(w==BIG) strcpy((p)->data,">");

if(w==SMALL) strcpy(p->data,"<");

if(w==EM) strcpy(p->data,"!=");

if(w==EQ) strcpy(p->data,"==");

if(w==ASSIGN) strcpy(p->data,"=");

p->lchild=(ASTTree)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->midchild=NULL;

p->lchild->rchild=NULL;

p->rchild=(ASTTree)malloc(sizeof(ASTnode));

p->rchild->lchild=NULL;

p->rchild->midchild=NULL;

p->rchild->rchild=NULL;

strcpy(p->lchild->data,token\_text0);

w=gettoken(fp);

strcpy(p->rchild->data,token\_text);

root->rchild=p;

w=gettoken(fp);

if(w==SEMI||w==RP) return root;

}

else if(MYprecede(lastc,token\_text)=='>'){

(root)->rchild=(ASTTree)malloc(sizeof(ASTnode));

(root)->rchild->lchild=NULL;

(root)->rchild->rchild=NULL;

(root)->rchild->midchild=NULL;

strcpy((root)->rchild->data,token\_text0);

p=(ASTTree)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->midchild=NULL;

p->rchild=NULL;

if(w==PLUS) strcpy(p->data,"+");

if(w==MINUS) strcpy(p->data,"-");

if(w==MULTIPLY) strcpy(p->data,"\*");

if(w==DIVIDE) strcpy(p->data,"/");

if(w==BIG) strcpy((p)->data,">");

if(w==SMALL) strcpy(p->data,"<");

if(w==EM) strcpy(p->data,"!=");

if(w==EQ) strcpy(p->data,"==");

if(w==ASSIGN) strcpy(p->data,"=");

p->lchild=(ASTTree)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->midchild=NULL;

p->lchild->rchild=NULL;

p->rchild=(ASTTree)malloc(sizeof(ASTnode));

p->rchild->lchild=NULL;

p->rchild->midchild=NULL;

p->rchild->rchild=NULL;

p->lchild=root;

w=gettoken(fp);

strcpy(p->rchild->data,token\_text);

root=p;

w=gettoken(fp);

if(w==SEMI||w==RP) return root;

}

else if(MYprecede(lastc,token\_text)=='='){

p=(ASTTree)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->midchild=NULL;

p->rchild=NULL;

if(w==PLUS) strcpy(p->data,"+");

if(w==MINUS) strcpy(p->data,"-");

if(w==MULTIPLY) strcpy(p->data,"\*");

if(w==DIVIDE) strcpy(p->data,"/");

if(w==BIG) strcpy((p)->data,">");

if(w==SMALL) strcpy(p->data,"<");

if(w==EM) strcpy(p->data,"!=");

if(w==EQ) strcpy(p->data,"==");

if(w==ASSIGN) strcpy(p->data,"=");

p->lchild=(ASTTree)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->midchild=NULL;

p->lchild->rchild=NULL;

p->rchild=(ASTTree)malloc(sizeof(ASTnode));

p->rchild->lchild=NULL;

p->rchild->midchild=NULL;

p->rchild->rchild=NULL;

strcpy(p->lchild->data,token\_text0);

w=gettoken(fp);

strcpy(p->rchild->data,token\_text);

root->rchild=p;

w=gettoken(fp);

if(w==SEMI||w==RP) return root;

}

}

return root;

}

#endif /\* expression\_h \*/

#ifndef function\_h

#define function\_h

ASTTree funcDef(FILE \*fp){//函数定义和声明

ASTTree root=NULL,p=NULL;

int w;

w=gettoken(fp);

(root)=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->lchild=NULL;

(root)->rchild=NULL;

(root)->midchild=NULL;

strcpy((root)->data,"函数");

w=gettoken(fp);

(root)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

root->lchild->lchild=NULL;

root->lchild->rchild=NULL;

root->lchild->midchild=NULL;

strcpy((root)->lchild->data,token\_text);

w=gettoken(fp);

(root)->midchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->lchild=NULL;

(root)->midchild->rchild=NULL;

(root)->midchild->midchild=NULL;

strcpy((root)->midchild->data,token\_text);

w=gettoken(fp);

//if(w!=LP) return NULL;

while(w!=RP){

w=gettoken(fp);

switch(w){

case INT:

(root)->midchild->midchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->midchild->lchild=NULL;

(root)->midchild->midchild->rchild=NULL;

(root)->midchild->midchild->midchild=NULL;

strcpy((root)->midchild->midchild->data,"形参类型");

(root)->midchild->midchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->midchild->lchild->lchild=NULL;

(root)->midchild->midchild->lchild->midchild=NULL;

(root)->midchild->midchild->lchild->rchild=NULL;

strcpy((root)->midchild->midchild->lchild->data,token\_text);

w=gettoken(fp);

(p)=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->rchild=NULL;

p->midchild=NULL;

strcpy(p->data,"形参名");

(root)->midchild->midchild->rchild=p;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((p)->lchild->data,token\_text);

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

//w=gettoken(fp);

/\*while(1){

if(w==SEMI)

{

break;

}

else if(w==COMMA){

w=gettoken(fp);

if(w!=IDENT)

{

printf("错误");

return NULL;

}

(q)=(ASTnode\*)malloc(sizeof(ASTnode));

q->lchild=NULL;

q->rchild=NULL;

q->midchild=NULL;

strcpy((q)->data,"形参");

(p)->rchild=q;

p=q;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

strcpy((p)->lchild->data,token\_text);

w=gettoken(fp);

}

else{

printf("错误");

return NULL;

}

}\*/

break;

case FLOAT:

(root)->midchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->lchild->lchild=NULL;

(root)->midchild->lchild->rchild=NULL;

(root)->midchild->lchild->midchild=NULL;

strcpy((root)->midchild->lchild->data,"形参类型");

(root)->midchild->lchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->lchild->lchild->lchild=NULL;

(root)->midchild->lchild->lchild->midchild=NULL;

(root)->midchild->lchild->lchild->rchild=NULL;

strcpy((root)->midchild->lchild->lchild->data,token\_text);

w=gettoken(fp);

(p)=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->rchild=NULL;

p->midchild=NULL;

strcpy(p->data,"形参名");

(root)->midchild->lchild->rchild=p;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((p)->lchild->data,token\_text);

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

/\*w=gettoken(fp);

while(1){

if(w==SEMI)

{

break;

}

else if(w==COMMA){

w=gettoken(fp);

if(w!=IDENT)

{

printf("错误");

return NULL;

}

(q)=(ASTnode\*)malloc(sizeof(ASTnode));

q->lchild=NULL;

q->rchild=NULL;

q->midchild=NULL;

strcpy((q)->data,"形参");

(p)->rchild=q;

p=q;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

strcpy((p)->lchild->data,token\_text);

w=gettoken(fp);

}

else{

printf("错误");

return NULL;

}

}\*/

break;

case CHAR:

(root)->midchild->rchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->rchild->lchild=NULL;

(root)->midchild->rchild->rchild=NULL;

(root)->midchild->rchild->midchild=NULL;

strcpy((root)->midchild->rchild->data,"形参类型");

(root)->midchild->rchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->midchild->rchild->lchild->lchild=NULL;

(root)->midchild->rchild->lchild->midchild=NULL;

(root)->midchild->rchild->lchild->rchild=NULL;

strcpy((root)->midchild->rchild->lchild->data,token\_text);

w=gettoken(fp);

(p)=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->rchild=NULL;

p->midchild=NULL;

strcpy(p->data,"形参名");

(root)->midchild->rchild->rchild=p;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((p)->lchild->data,token\_text);

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

break;

}

}

w=gettoken(fp);

(root)->rchild=(ASTnode\*)malloc(sizeof(ASTnode));

root->rchild->lchild=NULL;

(root)->rchild->rchild=NULL;

(root)->rchild->midchild=NULL;

if(w==SEMI){

strcpy((root)->rchild->data,"函数声明");

return root;

}

else{

if(w!=LPP) return NULL;

else{

strcpy((root)->rchild->data,"函数体");

(root)->rchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->rchild->lchild->lchild=NULL;

(root)->rchild->lchild->midchild=NULL;

(root)->rchild->lchild->rchild=NULL;

(root)->rchild->lchild=complex(fp);

}

}

return root;

}

#endif /\* function\_h \*/

#ifndef complex\_sentences\_h

#define complex\_sentences\_h

#include "expression.h"

ASTTree complex(FILE\*fp){//复合语句

int w;

ASTTree root = NULL;

ASTTree IF\_ELSE=NULL,MYwhile=NULL,MYfor=NULL;

(root)=(ASTnode\*)malloc(sizeof(ASTnode));

(root)->lchild=NULL;

(root)->rchild=NULL;

(root)->midchild=NULL;

(root)->rchild=(ASTnode\*)malloc(sizeof(ASTnode));//if-else

root->rchild->lchild=NULL;

root->rchild->rchild=NULL;

root->rchild->midchild=NULL;

strcpy(root->rchild->data,"if-else语句");

(root)->rchild->rchild=(ASTnode\*)malloc(sizeof(ASTnode));//return

root->rchild->rchild->lchild=NULL;

root->rchild->rchild->rchild=NULL;

root->rchild->rchild->midchild=NULL;

strcpy(root->rchild->rchild->data,"return语句");

(root)->rchild->rchild->rchild=(ASTnode\*)malloc(sizeof(ASTnode));//while

root->rchild->rchild->rchild->lchild=NULL;

root->rchild->rchild->rchild->rchild=NULL;

root->rchild->rchild->rchild->midchild=NULL;

strcpy(root->rchild->rchild->rchild->data,"while语句");

(root)->rchild->rchild->rchild->rchild=(ASTnode\*)malloc(sizeof(ASTnode));//for

root->rchild->rchild->rchild->rchild->lchild=NULL;

root->rchild->rchild->rchild->rchild->rchild=NULL;

root->rchild->rchild->rchild->rchild->midchild=NULL;

strcpy(root->rchild->rchild->rchild->rchild->data,"for语句");

(root)->rchild->rchild->rchild->rchild->rchild=(ASTnode\*)malloc(sizeof(ASTnode));//expression

root->rchild->rchild->rchild->rchild->rchild->lchild=NULL;

root->rchild->rchild->rchild->rchild->rchild->rchild=NULL;

root->rchild->rchild->rchild->rchild->rchild->midchild=NULL;

strcpy(root->rchild->rchild->rchild->rchild->rchild->data,"表达式");

(IF\_ELSE)=(ASTnode\*)malloc(sizeof(ASTnode));

(IF\_ELSE)->lchild=NULL;

(IF\_ELSE)->rchild=NULL;

(IF\_ELSE)->midchild=NULL;

strcpy((IF\_ELSE)->data,"if-else");

MYwhile=(ASTnode\*)malloc(sizeof(ASTnode));

MYwhile->lchild=NULL;

MYwhile->rchild=NULL;

MYwhile->midchild=NULL;

strcpy((MYwhile)->data,"while");

MYfor=(ASTnode\*)malloc(sizeof(ASTnode));

MYfor->lchild=NULL;

MYfor->rchild=NULL;

MYfor->midchild=NULL;

strcpy((MYfor)->data,"for");

strcpy((root)->data,"复合语句");

w=gettoken(fp);

w=gettoken(fp);

while(w!=RPP){

if(w>=INT&&w<=FLOAT){

root->lchild=ExtVarDef1(fp);

}

else{

//w=gettoken(fp);

//printf("%s",token\_text);

switch(w){

case IF:

w=gettoken(fp);

//printf("%s",token\_text);

if(w!=LP) return NULL;

(IF\_ELSE)->lchild=MYexp(fp);

(IF\_ELSE)->midchild=MYexp(fp);

w=gettoken(fp);

w=gettoken(fp);

if(w!=ELSE) return NULL;

(IF\_ELSE)->rchild=MYexp(fp);

(root)->rchild->lchild=IF\_ELSE;

w=gettoken(fp);

//w=gettoken(fp);

break;

case RETURN:

(root)->rchild->rchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));//return

root->rchild->rchild->lchild->lchild=NULL;

root->rchild->rchild->lchild->rchild=NULL;

root->rchild->rchild->lchild->midchild=NULL;

strcpy(root->rchild->rchild->lchild->data,"return");

//w=gettoken(fp);

(root)->rchild->rchild->lchild->lchild=(ASTnode\*)malloc(sizeof(ASTnode));//return

root->rchild->rchild->lchild->lchild->lchild=NULL;

root->rchild->rchild->lchild->lchild->rchild=NULL;

root->rchild->rchild->lchild->lchild->midchild=NULL;

root->rchild->rchild->lchild->lchild=MYexp(fp);

w=gettoken(fp);

break;

case WHILE:

w=gettoken(fp);

//printf("%s",token\_text);

if(w!=LP) return NULL;

MYwhile->lchild=(ASTTree)malloc(sizeof(ASTnode));

MYwhile->lchild->lchild=NULL;

MYwhile->lchild->rchild=NULL;

MYwhile->lchild->midchild=NULL;

(MYwhile)->lchild=MYexp(fp);

w=gettoken(fp);

if(w==SEMI){

MYwhile->midchild=(ASTTree)malloc(sizeof(ASTnode));

MYwhile->midchild->lchild=NULL;

MYwhile->midchild->rchild=NULL;

MYwhile->midchild->midchild=NULL;

strcpy(MYwhile->midchild->data,"死循环");

}

else if(w==LPP){

MYwhile->rchild=(ASTTree)malloc(sizeof(ASTnode));

MYwhile->rchild->lchild=NULL;

MYwhile->rchild->rchild=NULL;

MYwhile->rchild->midchild=NULL;

strcpy(MYwhile->rchild->data,"循环体");

MYwhile->rchild->lchild=complex(fp);

}

(root)->rchild->rchild->rchild->lchild=MYwhile;

break;

case FOR:

w=gettoken(fp);

//printf("111%s\n",token\_text);

if(w!=LP) return NULL;

MYfor->lchild=(ASTTree)malloc(sizeof(ASTnode));

MYfor->lchild->lchild=NULL;

MYfor->lchild->rchild=NULL;

MYfor->lchild->midchild=NULL;

MYfor->lchild->lchild=MYexp(fp);

MYfor->lchild->midchild=MYexp(fp);

MYfor->lchild->rchild=MYexp(fp);

w=gettoken(fp);

if(w==SEMI){

MYfor->midchild=(ASTTree)malloc(sizeof(ASTnode));

MYfor->midchild->lchild=NULL;

MYfor->midchild->rchild=NULL;

MYfor->midchild->midchild=NULL;

strcpy(MYfor->midchild->data,"死循环");

}

else if(w==LPP){

MYfor->rchild=(ASTTree)malloc(sizeof(ASTnode));

MYfor->rchild->lchild=NULL;

MYfor->rchild->rchild=NULL;

MYfor->rchild->midchild=NULL;

strcpy(MYfor->rchild->data,"循环体");

MYfor->rchild->lchild=complex(fp);

}

(root)->rchild->rchild->rchild->rchild->lchild=MYfor;

break;

case EXP:

root->rchild->rchild->rchild->rchild->rchild->lchild=MYexp(fp);

break;

//w=gettoken(fp);

}

}

w=gettoken(fp);

}

return root;

}

#endif /\* complex\_sentences\_h \*/

#ifndef variable\_h

#define variable\_h

ASTTree pre(FILE \*fp){

int w;

ASTTree root=NULL;

root=(ASTTree)malloc(sizeof(ASTnode));

root->lchild=(ASTTree)malloc(sizeof(ASTnode));

root->lchild->lchild=NULL;

root->lchild->midchild=NULL;

root->lchild->rchild=NULL;

root->midchild=(ASTTree)malloc(sizeof(ASTnode));

root->midchild->lchild=NULL;

root->midchild->midchild=NULL;

root->midchild->rchild=NULL;

root->rchild=(ASTTree)malloc(sizeof(ASTnode));

root->rchild->lchild=NULL;

root->rchild->midchild=NULL;

root->rchild->rchild=NULL;

strcpy(root->data,"编译预处理");

w=gettoken(fp);

if(w!=EXP) return NULL;

strcpy(root->lchild->data,"#");

w=gettoken(fp);

if(w!=INCLUDE) return NULL;

strcpy(root->midchild->data,"include");

w=gettoken(fp);

if(w!=SMALL) return NULL;

w=gettoken(fp);

if(w!=hfile) return NULL;

strcpy(root->rchild->data,token\_text);

w=gettoken(fp);

if(w!=BIG) return NULL;

w=gettoken(fp);

return root;

}

ASTTree ExtVarDef(FILE \*fp){//外部变量

char token\_text0[100];

int w;

ASTTree root=NULL,p=NULL,q=NULL;

root=(ASTTree)malloc(sizeof(ASTnode));

root->midchild=NULL;

strcpy((root)->data,"外部变量类型：");

w=gettoken(fp);

strcpy(token\_text0,token\_text);

(root)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((root)->lchild->data,token\_text0);

root->lchild->lchild=NULL;

root->lchild->rchild=NULL;

root->lchild->midchild=NULL;

(p)=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->rchild=NULL;

p->midchild=NULL;

(root)->rchild=p;

strcpy((p)->data,"外部变量名：");

w=gettoken(fp);

strcpy(token\_text0,token\_text);

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((p)->lchild->data,token\_text0);

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

w=gettoken(fp);

while(1){

if(w==SEMI)

{

return root;

}

else if(w==COMMA){

w=gettoken(fp);

if(w!=IDENT)

{

//printf("错误");

return NULL;

}

(q)=(ASTnode\*)malloc(sizeof(ASTnode));

q->lchild=NULL;

q->rchild=NULL;

q->midchild=NULL;

strcpy((q)->data,"外部变量名：");

(p)->rchild=q;

p=q;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

strcpy((p)->lchild->data,token\_text);

w=gettoken(fp);

}

else{

//printf("错误");

return NULL;

}

}

}

ASTTree ExtVarDef1(FILE \*fp){//外部变量

char token\_text0[100];

int w;

ASTTree root=NULL,p=NULL,q=NULL;

root=(ASTTree)malloc(sizeof(ASTnode));

root->midchild=NULL;

strcpy((root)->data,"局部变量类型：");

strcpy(token\_text0,token\_text);

(root)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((root)->lchild->data,token\_text0);

root->lchild->lchild=NULL;

root->lchild->rchild=NULL;

root->lchild->midchild=NULL;

(p)=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild=NULL;

p->rchild=NULL;

p->midchild=NULL;

(root)->rchild=p;

strcpy((p)->data,"局部变量名：");

w=gettoken(fp);

strcpy(token\_text0,token\_text);

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

strcpy((p)->lchild->data,token\_text0);

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

w=gettoken(fp);

while(1){

if(w==SEMI)

{

return root;

}

else if(w==COMMA){

w=gettoken(fp);

if(w!=IDENT)

{

//printf("错误");

return NULL;

}

(q)=(ASTnode\*)malloc(sizeof(ASTnode));

q->lchild=NULL;

q->rchild=NULL;

q->midchild=NULL;

strcpy((q)->data,"局部变量名：");

(p)->rchild=q;

p=q;

(p)->lchild=(ASTnode\*)malloc(sizeof(ASTnode));

p->lchild->lchild=NULL;

p->lchild->rchild=NULL;

p->lchild->midchild=NULL;

strcpy((p)->lchild->data,token\_text);

w=gettoken(fp);

}

else{

//printf("错误");

return NULL;

}

}

}

#endif /\* variable\_h \*/

#ifndef Tree\_function\_h

#define Tree\_function\_h

int Visit(ASTTree T){

if(T==NULL) return -5;

printf("%s \n",T->data);

return -6;

}

int Pre\_order(ASTTree T, int (\*Visit)(ASTTree T)) {

if (T) {

if (Visit(T))

if (Pre\_order(T->lchild, Visit))

if (Pre\_order(T->midchild, Visit))

if (Pre\_order(T->rchild, Visit)) return -6;

return -5;

}

else return -6;

}

#endif /\* Tree\_function\_h \*/

#include"预定义常量.h"

#ifndef \_\_\_\_\_h

#define \_\_\_\_\_h

int gettoken(FILE \*fp){

memset(token\_text,0, sizeof(token\_text));

char c;

int i,j=0;

int count=0;

c=fgetc(fp);

while(1){

if(c==' ') c=fgetc(fp);

else break;

}

if((c>='A'&&c<='Z')||(c>='a'&&c<='z')||(c=='\_')){

do{

token\_text[j++]=c;

c=fgetc(fp);

if(c=='.')

{

token\_text[j++]=c;

c=fgetc(fp);

if(c=='h')

{token\_text[j++]=c; return hfile;}

else return ERROR\_TOKEN;

}

}while((c>='A'&&c<='Z')||(c>='a'&&c<='z')||(c>='0'&&c<='9'));

ungetc(c,fp);

for(i=0;i<11;i++){

if(strcmp(keywords[i],token\_text)==0)

return i;

}

return IDENT;

}

if(c>='0'&&c<='9'){

do{

token\_text[j++]=c;

c=fgetc(fp);

if(c=='x')

{token\_text[j++]=c;

c=fgetc(fp);

if(c<='0'||c>='9') return ERROR\_TOKEN;

}

else if(c>='a'&&c<='z') return ERROR\_TOKEN;

}while((c>='0'&&c<='9'));

ungetc(c,fp);

return INT\_CONST;

}

switch (c) {

case '=':c=fgetc(fp);

if(c=='=') return EQ;

ungetc(c,fp);

return ASSIGN;

case '+':c=fgetc(fp);

if(c=='+') return PLUSPLUS;

ungetc(c,fp);

return PLUS;

case '-':c=fgetc(fp);

if(c=='-') return MINUSMINUS;

ungetc(c,fp);

return MINUS;

case '\*':

return MULTIPLY;

case '/':

return DIVIDE;

case ';':

return SEMI;

case ',':

return COMMA;

case '%':

return MOD;

case '&':c=fgetc(fp);

if(c=='&') return AND;

ungetc(c,fp);

return ERROR\_TOKEN;

case '|':c=fgetc(fp);

if(c=='|') return OR;

ungetc(c,fp);

return ERROR\_TOKEN;

case '(':

return LP;

case ')':

return RP;

case '\n':count++;

break;

case '>':return BIG;

case '<':return SMALL;

case '{':return LPP;

case '}':return RPP;

case '#':return EXP;

case '!':c=fgetc(fp);

if(c=='=') return EM;

else return ERROR\_TOKEN;

default:if(feof(fp)) return EOF;

else return ERROR\_TOKEN;

}

return 100;

}

#endif /\* \_\_\_\_\_h \*/

#ifndef \_\_\_\_\_\_h

#define \_\_\_\_\_\_h

char keywords[11][11]={"int","char","float","while","if","else","break","for","return","continue","include"

};

enum token\_kind{INT=0,CHAR,FLOAT,WHILE,IF,ELSE,BREAK,FOR,RETURN,CONTINUE,INCLUDE,hfile,ERROR\_TOKEN,IDENT,INT\_CONST,EQ,PLUS,PLUSPLUS,MINUS,MINUSMINUS,MULTIPLY,DIVIDE,ASSIGN,SEMI,COMMA,MOD,AND,OR,LP,RP,BIG,SMALL,EM,LPP,RPP,EXP};

char token\_text[100];

typedef struct ASTnode{

char data[100];

struct ASTnode \*lchild;

struct ASTnode \*rchild;

struct ASTnode \*midchild;

}ASTnode,\*ASTTree;

typedef struct stack{

int \*base;

int \*top;

}stack;

#endif /\* \_\_\_\_\_\_h \*/

附录二 程序使用说明

1. 在文本编辑器里编写一段代码
2. 格式为：外部变量定义

函数定义

外部变量定义

1. 一句表达式中最多三个运算符
2. 表达式之前必须加上#，结尾不需要分号
3. 所有大括号需要另起一行
4. 正常使用，除上述情况外，语句需要满足C语言规范
5. 语句顺序不固定
6. 附上一段正常测试用例：

int a,b;

int fun(int a;float b){

if(a>b) m=a;

else m=b;

while(a>b){

if(a>b) x=a;

else x=b;

while(c>d);

for(j=0;j<9;j++);

# a\*b+10

return x;

}

for(i=0;i<9;i++);

int a,b;

# a+b\*10

return m;

}