

**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： cs2003**

**学 号： U202015360**

**姓 名： 胡沁心**

**指导教师： 卢萍**

**报告日期： 2021/10/14**

**计算机科学与技术学院**

# 任务书

## 设计内容

在计算机科学中，抽象语法树（abstract syntax tree或者缩写为AST），是将源代码的语法结构的用树的形式表示，树上的每个结点都表示源程序代码中的一种语法成分。之所以说是“抽象”，是因为在抽象语法树中，忽略了源程序中语法成分的一些细节，突出了其主要语法特征。

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。

在《高级语言源程序格式处理工具》这个题目中，首先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。

## 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。

## 参考文献

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

[4] 殷立峰. Qt C++跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197

[5] 严蔚敏等.数据结构题集（C语言版）.清华大学出版社

# 目录

[任务书 I](#_Toc82984156)

[设计内容 I](#_Toc82984157)

[设计要求 I](#_Toc82984158)

[参考文献 II](#_Toc82984159)

[1 引言 4](#_Toc82984161)

[1.1 课题背景与意义 4](#_Toc82984162)

[1.1.1背景 4](#_Toc82984163)

[1.1.2意义 4](#_Toc82984164)

[1.2 国内外研究现状 4](#_Toc82984165)

[1.3 课程设计的主要研究工作 5](#_Toc82984166)

[2 系统需求分析与总体设计 6](#_Toc82984167)

[2.1 系统需求分析 6](#_Toc82984168)

[2.2 系统总体设计 6](#_Toc82984169)

[3 系统详细设计 8](#_Toc82984170)

[3.1 数据结构的定义 8](#_Toc82984171)

[3.2 主要算法设计 11](#_Toc82984172)

[4 系统实现与测试 17](#_Toc82984173)

[4.1 系统实现 17](#_Toc82984174)

[4.2 系统测试 36](#_Toc82984175)

[5 总结与展望 58](#_Toc82984176)

[5.1 全文总结 58](#_Toc82984177)

[5.2 工作展望 58](#_Toc82984178)

[6 体会 58](#_Toc82984179)

# 1 引言

## 1.1 课题背景与意义

抽象语法树（Abstract Syntax Tree，AST），或简称语法树（Syntax tree），是源代码语法结构的一种抽象表示。

## 1.1.1 背景

抽象语法树以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构。之所以说语法是“抽象”的，是因为这里的语法并不会表示出真实语法中出现的每个细节。

### 1.1.2 意义

和抽象语法树相对的是具体语法树（通常称作分析树）。一般的，在源代码的翻译和编译过程中，语法分析器创建出分析树，然后从分析树生成AST。一旦AST被创建出来，在后续的处理过程中，比如语义分析阶段，会添加一些信息。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 国内外研究现状

（1）AST生成工具:

Esprima：解析器；Acorn：esprima后的轮子,目前使用最多。webpack也使用此工具；Astexplorer：在线生成工具；Espree：最初从esprima fork出来，来自eslint，用于eslint；babel-parser：原babylon，最初从acorn fork出来；UglifyJS2：自带parser；shift-parser-js：自己定义了一套AST规范shift-spec。

（2）可以使用的插件工具:

esprima：code=>AST代码转AST；estraverse：traverse AST转换树；escodegen：AST=>code；

（3）基于AST的工具

Eslint； Webpack；UglifyJS：代码压缩；Prettier：使用@babel/parser、angular-estree-parser；Typescript：自带parser、transformer、codeGenerator；代码重构：1.Recast；2.jscodeshift；3.react-codemod；其他，例如：IDE错误提示、格式化、高亮、自动补全等

（4）AST转换工具

estraverse

（5）代码生成工具：

escodegen

## 1.3 课程设计的主要研究工作

1.了解AST的概念、作用和流程。

2.了解各种AST生成网站和软件。

3.阅读网络上不同作者写的详细AST解说，并使用生成AST的网站，了解AST呈现的形式。

4.了解编译器的运行原理，报错和警告的流程

# 2 系统需求分析与总体设计

## 2.1 系统需求分析

1.读取c代码,识别单词,包括要求中所有关键词及其语句，标识符，界定符，运算符，文件包括，宏定义以及注释的读取和识别。其中能够识别的数字类型有：十进制，带后缀的十进制，八进制，十六进制，小数。支持负数识别和括号运算。

2.对不合法字符、不合法字符串进行报错，并将所有单词及其编码写入文件中。

3.分析语法，进行语法报错，其中包括：

1)所有界定符和运算符的错误使用

2)重复定义或未定义的变量和函数

3)没有if的else和else-if语句

4)break、continue、return语句的错误使用

5)函数调用时参数错误

6)变量赋值类型错误

7)不合法的宏定义命名和变量命名

8)对没有效果的语句输出警告

4.根据测试代码生成AST

5.遍历生成的AST，输出AST内容

6.根据AST编排缩进，写入新的c文件

## 2.2 系统总体设计

(1)功能模块(每个文件对应一个模块)

表2-1 功能模块说明表

|  |  |  |
| --- | --- | --- |
| 文件 | 功能模块 | 功能 |
| main.cpp | 主函数 | 文件的打开、关闭，调用各个模块等各项基本指令 |
| read.h | 词法分析 | 读取文件内容，获得字符串和符号的类型编号，对非法字符报错 |
| judge.h | 判别类型 | 判断读取的字符串的类型。包括字符类型，关键字类型，数字类型，其中数字类型包括十进制，带后缀的十进制，八进制，十六进制，小数。判断字符串中字符的合法性 |
| compile.h | 语法分析 | 子程序的选择和递归调用，所有语句的处理，编译预处理，变量的声明，函数的声明和定义，for，while，if-else，if-else-if，return，break，continue语句，行注释，块注释，条件语句，赋值语句的处理并建AST |
| tree.h | 建AST | AST结点的新建和插入 |
| error.h | 报错 | 所有语法和字符的判错报错，以及部分语句的警告 |
| traverse.h | 输出AST | 中序遍历AST，编排缩进，输出内容 |
| save.h | 源文件编排 | 遍历AST，编排缩进，写入源程序文件中 |



图2-1 系统模块结构图

# 3 系统详细设计

## 3.1 数据结构的定义

（1）需要处理的数据

表3-1 系统数据

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据 | 类型 | 名字 | 说明 | | | | | |
| 宏定义，函数，变量总数 | int | n |  | | | | | |
| 当前所在的字符的位置 | int | i |  | | | | | |
| 测试用例内容 | string | s | 下标为i | | | | | |
| 当前读取的字符串 | string | t |  | | | | | |
| 读取的字符串长度 | int | j |  | | | | | |
| 读取的前一字符串 | string | pr |  | | | | | |
| 读取的前一个符号的编号 | int | pre |  | | | | | |
| 标记是否已进入下一嵌套 | bool | c |  | | | | | |
| 语句嵌套层数 | int | nest |  | | | | | |
| 循环层数 | int | loop |  | | | | | |
| 当前行数 | int | l |  | | | | | |
| 圆括号层数 | int | paren |  | | | | | |
| 方括号层数 | int | bracket |  | | | | | |
| 当前所在函数序号 | int | temp |  | | | | | |
| 注释 | string | note |  | | | | | |
| 符号类型编号 | enum | token\_k |  | | | | | |
| 字符串类型编号 | enum | string\_k |  | | | | | |
| 报错类型编号 | enum | error\_k |  | | | | | |
| 当前的父亲结点 | Node\* | T |  | | | | | |
| 每层嵌套所在的父亲节点 | Node\*[10] | f | 下标为nest | | | | | |
| 不同优先级运算符的父亲节点 | Node\*[5] | r | 优先级 | 0 | 1 | 2 | 3 | 4 |
| 符号 | = | &&，|| | <,>,==,!= | +,- | \*,/,% |
| 声明过的宏定义、变量及函数的信息 | variable | st | 下标为n | | | | | |
| 当前读取的字符的编号 | dfa | op |  | | | | | |

表3-2 系统数据结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据 | 数据类型 | 数据项 | 数据项类型 | 名字 |
| 声明过的宏定义、变量及函数的信息 | struct  (variable) | 变量数据类型或函数返回值类型 | int | type |
| 变量或函数名 | char | name |
| 函数参数类型 | int | param\_type |
| 函数参数名字 | char | param\_name |
| 数组变量大小 | char | array\_size |
| AST结点数据结构 | struct  (Node) | 结点类型 | char | key1 |
| 具体信息 | char | key2 |
| 孩子数量 | int | sum |
| 孩子结点 | Node\*[10] | child |
| 读取的字符的编号 | struct  (dfa) | 字符串编号 | int | s |
| 字符编号 | int | t |



图3-1 系统数据联系图1



图3-2 系统数据联系图2

## 3.2 主要算法设计

1.主程序模块

流程：



图3-3 主程序流程图

2.读取模块

算法思想：递归

流程：先读取一个字符串和一个字符，分别获取它们的类型编号

1)DFA（识别读取的字符串和符号）：



图3-4 DFA流程图

2)注释读取：当读到注释界定符时，进入编译模块，读取行注释和块注释，写入AST中，读到换行符或界定符时结束

3.判别模块

流程：四个函数分别实现四个功能。

1)数字类型判别

2)字符判别

3)关键字判别

4)字符串合法字符判别

前三个函数的返回值为当前读取的字符串的类型编号。第四个函数若返回值为0，结束文件的当前字符串读取，否则继续读取文件。

4.编译模块

算法思想：过程型递归（子程序下降递归）

流程：分为两类语句，含关键字的语句和含运算符的语句

1)关键字语句部分：根据获取的字符串编号，通过switch语句选择相应的子函数进行递归调用，直到文件结束。其中读到if语句、for语句、while语句的条件语句或赋值语句时，调用运算符处理程序。return语句若有返回值，读完关键字后调用运算符处理程序。



图3-5 关键字语句部分

2)条件语句，赋值语句，函数调用部分：处理语句的运算符，根据运算符的优先级建AST。

1.运算符处理程序的作用：判断当前读取的字符串是否是函数，如果是，调用函数调用程序，否则将当前字符串写入ASR，并进行递归中转

2.递归中转程序的作用：插入运算符结点，更新所有运算符的父亲结点，读取下一字符串和运算符或分隔符。最后调用运算符处理程序，实现递归中转。

3.函数调用程序的作用：处理函数调用，将函数调用部分建入AST中，并对函数参数进行判错。



图3-6 含运算符的语句部分

5.建AST模块

流程：分为两种情况，由编译模块的函数选择调用

1)在当前结点下创建新的孩子结点

2)在当前结点和该结点的最后一个孩子结点之间插入新结点

6.报错模块

流程：根据编译模块模块的指令运行，七个函数实现七个功能：

1. 变量或函数未定义判定
2. 变量或函数重复定义判定
3. 变量与函数命名合法性判定
4. 宏定义命名合法性判定
5. 符号判错
6. 合法字符串判断
7. 其他语法报错(switch语句选择报错类型，包括变量赋值类型报错，函数参数类型报错，无效果语句警告，break、continue、return语句报错等)

7.输出AST模块

算法思想：结构性递归

流程：先根遍历AST，根据结点的深度编排缩进，输出结点中的内容

8.源文件编排模块

算法思想：结构性递归

流程：从根节点开始先根遍历，遇到条件语句、赋值语句、函数调用、含返回值的返回语句等可能含有运算符的语句，改用中序遍历，含运算符的语句结束后，继续先根遍历。同时根据语句嵌套的层数编排缩进，写入源文件程序中。

# 4 系统实现与测试

## 4.1 系统实现

硬件环境：CPU：i7-1165G7，内存：512MB，硬盘：512GB，显卡：2G

软件环境：Windows xp操作系统，Code::Blocks 10.05

根据3.1的设计，用C语言定义各种数据类型：

|  |
| --- |
| typedef struct variable{//变量信息储存  int type; //数据类型或函数返回类型  char name[100]; //变量或函数名  int param\_type; //函数参数类型  char param\_name[100]; //函数参数名字  char array\_size[100]; //数组变量大小  }variable;  typedef struct Node{ //语法树结点结构  char key1[100]; //类型  char key2[100]; //具体信息  int sum; //孩子数量  Node\* child[10]; //孩子们  }Node,\*Tree;  typedef struct dfa{ //字符出和符号的类型编号结构  int s,t;  }dfa; |

函数说明：

1.主程序部分

函数：int main()，执行基本指令，进入其他模块

流程：

1. 新建整个语法树的根节点，将测试文件读入字符串s中。
2. 依次进入编译模块，遍历模块和源程序编排模块。



图4-1 主程序流程图

2.读取模块

函数：

1)void skip()：跳过无意义的字符

流程：跳过连续的空格、\t、\r、\n。读到\n时行数l加一

2)void read()：读取文件，获取类型编号

流程：

1. 按DFA读取一个字符串和一个字符
2. 如果字符串的第一个字符是减号，继续读取
3. 若读到注释界定符，进入编译模块，调用delimit函数，将注释写进AST中。
4. 用switch语句选择，分别获取字符串和字符的类型编号。若有不合法的符号，进行报错。



图4-2 文件读取

3.judge(识别模块)

包括函数：

1)int num\_judge(int jj)：数字类型识别，jj为字符串长度

流程：

1. 如果读取的字符串中，第一个字符为‘-’，即读到的是负数，去掉字符串中的符号，字符串长度减一。
2. 如果字符串长度为1且字符为0~9，那么就是int型
3. 否则继续识别，流程如下图



图4-3 数字类型识别

2)int char\_judge()：字符识别

流程：若首个字符和末位字符为单引号，则返回char型，否则返回0

3)int key\_judge()：关键字识别

流程：根据字符串的值，返回相应的关键字的编号，不是关键字就返回0

4) int judge()：字符串合法字符识别

流程：判断当前读取的字符是否为减号、数字、字母、下划线、小数点或单引号，不是则返回0，否则返回1

4.编译模块

包括函数：

1)int subroutine()：子函数选择和调用程序

流程：

1. 进入读取模块，读取文件内容，获得类型编号
2. 更新父亲结点
3. 若读到‘}’时返回0，表示复合语句块结束。
4. 若文件结束，返回1。
5. 否则根据获取的字符串编号，通过switch语句选择子函数进行递归调用。



图4-4 子程序调用

2)void compile()：总编译指令

流程：while循环调用subroutine函数。当返回值为0时，代表复合语句块结束，返回原函数，否则继续进行while循环，直到文件结束。

3)void init(int type)：将新定义的宏定义或变量或函数存入st中。

流程：

1. 数据类型赋值为type
2. 数据名赋值为t
3. 参数类型初始化为0。

4)void delimit()：读取注释。

流程：

1. 若为行注释，读到\n结束。若为块注释，读到\n行数l加一，读到\*/结束。
2. 读取的字符串存入note，写入AST中。

5)void def\_pre()：预处理，文件包含和宏定义

流程：



图4-5 预处理子程序

6)void def\_var(int type)：变量，函数的声明和定义子程序。

type为关键字的数据类型编号

流程：



图4-6 声明语句子程序

7) void command()：return，break，continue语句处理

流程：

1. 识别关键字
2. 若是有返回值的return语句，调用运算符处理程序



图4-7 命令语句子程序

8) void go\_loop(int type)：循环语句处理

type为该语句的关键字的类型编号

流程：

1. 循环层数和嵌套层数分别加一
2. 识别关键字，调用相应函数
3. 复合语句判断，进入下一层嵌套
4. 语句结束后，循环层数和嵌套层数分别减一



图4-8 循环语句子程序

9) void go\_for()：for语句处理

流程：读取括号中的三个字句，依次为赋值语句，条件语句，赋值语句，若语句不为空，即j!=0，调用var(T,-1)进行处理。否则跳过，处理下一个语句。每读完一个子句，判别语句是否有效果和符号是否有错并报错。

10) void go\_while()：while语句处理

流程：

1. 读取括号中的条件语句
2. 若语句为空，报错，否则调用var(T,-1)进行条件语句处理。

11) void compound\_judge()：判断是否是复合语句并进行递归之前的部分操作

流程：

1. 判断是否是复合语句
2. 进入编译模块，如果为复合语句，调用总编译函数，否则调用子程序递归函数。
3. c赋值为0，表示离开嵌套语句。



图4-9 复合语句判断

12) void compounds(const char\* r)：处理for，while，if的语句嵌套

流程：

1. 字符串r为“expression”或“compound”，表示单个语句或复合语句，作为要写入AST的信息。
2. 进入建AST模块，调用CreateNode函数新建结点，新结点作为新的父亲结点
3. 嵌套层数nest加一。
4. c赋值为1，表示进入嵌套语句。

13) void reassign(int x)：更新父亲结点

流程：for循环语句，将优先级比x低的运算符的父亲节点更新为当前的父亲结点T。

14) recursion1(int x,int y)：递归中转程序1，针对赋值运算符和括号运算符的递归中转。

x为需要更新的父亲结点的最高优先级，y为当前运算符的优先级

流程：

1. 先更新所有运算符的父亲结点
2. 然后读取下一个要处理的字符串和运算符
3. 进行下一次递归。



图4-10 递归1

15) recursion2(int x, const char\* tt) ：递归中转程序2，针对算数运算符和关系运算符的递归中转。

x为当前运算符的优先级，tt为要写入AST的运算符。

流程：

1. 先在AST中插入当前的运算符
2. 更新所有运算符的父亲结点
3. 然后读取下一个要处理的字符串和运算符
4. 进行下一次递归。



图4-11 递归2

16)void func(Tree root)：函数调用语句处理。

root为建该调用函数结点的根节点。

流程：

1. 先更新父亲结点
2. 判断该函数调用是否有参数，没有就报错返回
3. 然后读取函数参数，判断参数的个数是否错误。
4. 继续递归。



图4-13 函数调用处理

17) void assign()：对“+=”“\*=”类型符号的处理

例如j+=i，将算式转化为j=j+i。上一个字符串已经保存在pr中。

流程：

1. 将新的数和赋值运算符加入AST
2. 将pr中的字符串加入AST中
3. 最后将赋值运算符的两个结点调换顺序。

示意图如下



图4-12 算式处理示意图

18)void oprator(Tree root,int x)：运算符处理。

root为根节点指针。若x>=0，则为上一个运算符的优先级，否则作0处理。

流程：

1. 根据x的值选择建AST的根节点，若x>=0，T=root，否则T=root的最后一个孩子。
2. 判别字符串编号op.s。若不是标识符或数字或字符，报错返回。若是函数名，调用func(T)，否则建AST。
3. Str赋值为为运算符字符串。
4. 判别字符编号op.t。若读到’}’、圆括号数paren<0、方括号数bracket<0，报错并返回。
5. 读取其他合法符号的操作具体如下图：



图4-14 运算符处理

5.tree(建树模块)

函数包括：

1)void CreateNode(const char \*t1, const char \*t2)：在当前结点下创建新的孩子结点

流程：

1. 新建结点p，作为T的第sum个结点
2. t1写入p->key1，t2写入p->key2
3. T的孩子总数T->sum加一

新结点结构如下：



图4-15 结点数据结构

2)void InsertNode(Tree root,const char\* str,int x)：在两个结点中插入结点

x为标记数字。

流程：

1. 新建结点p
2. x为单数则将字符串写入key1，双数则写入key2。当x=3时，为“+=”类型的运算符，key2写入一个空格作为标记。

示意图如下



图4-16 插入结点示意图

9.error(报错模块)

包括函数函数：

1)int punc (int x)：标点报错

流程：若op.t!=x，返回0，若op.t不是’;’，退回一个字符。

2)int redeclare\_judge()：重复定义判别

流程：遍历st，将每个变量或函数名与当前字符串进行比较，判断是否重复定义字符串t，不是返回1，否则返回0

3)int var\_judge()：已定义变量判别

流程：

1. 遍历st，将每个变量或函数名与当前字符串进行比较，判断字符串t是否是已定义的变量或函数
2. 将当前所在的函数的参数与字符串相比较，判断是否相同
3. 如果都不相同返回0，否则返回1

4)void declare\_judge()：字符串合法性判别

流程：

1. 调用判别模块中的函数，判断字符串t是否是变量，关键字，数字或字符，若都不是，报错。
2. 若判断为char类型，如果界定符之间的字符个数大于1，警告。

5)void name\_judge()：判断变量或函数的命名是否合法

流程：

1. 判断首字母是否为数字
2. 判断是否含有除数字、大小写字母、下划线外的其他符号，若有，报错。

6)void macroname\_judge()：判断宏定义的命名是否合法

流程：判断是否有除大写字母、下划线外的其他符号，若有，报错。

7)void error(int x)：其他语法报错，x为报错类型的编号

流程：根据x的值，通过switch语句输出不同类型的报错或警告信息。

10.输出AST模块

函数：void PreOrderTraverse(Tree T,int d)

流程：先根遍历AST，根据结点的深度编排缩进，输出结点中key1和key2字符串的内容

11.源文件编写模块

包括函数：

1) void indent(int d)：按照语句嵌套层数编排缩进，d为当前所在的层数

流程：若d>1，输出(d-1)\*4个空格

2) void next(Tree T,int d)：遍历T的所有孩子节点

T为当前所在的结点，d为当前所在的层数

流程：for循环语句，按顺序调用save函数，遍历T的每一个孩子结点

3) void InOrderTraverse(Tree T)：输出条件语句和赋值语句

T为当前所在的结点

流程：

以T为根节点进行中序遍历，输出结点内容：

1. 如果内容为”assign”，输出’=’
2. 如果内容为”=”，输出’==’
3. 如果内容为”paren”，输出左右圆括号
4. 如果内容为”bracket”，输出左右方括号
5. 其他按原样输出。

4) void save(Tree T,int d)：先序遍历，输出语句。

流程：先后对key1和key2进行判别，进行相应操作。具体如下图



图4-17 生成源程序文件(key1)



图4-18 生成源程序文件(key2)



图4-19 生成源程序文件(其它)

由于编译模块递归算法运用多，函数调用关系复杂，且读取模块、报错模块和建树模块的函数调用次数很多，画入图中会过于混乱，所以部分关系不在函数调用图中显示。下图为去除部分读取模块调用关系、报错模块和建树模块的函数调用关系图，但可以展示程序的主要结构和主要运行过程。



图4-20 函数调用关系图

## 4.2 系统测试

软件测试方法：

1)静态测试：分析或检查源程序的语句、结构、过程等来检查程序是否有错误

2)动态测试：构造测试实例，执行程序并分析结果

一共选取了三个模块进行测试：

1.编译模块

（1）功能与设计目标

语法结构分析与生成抽象语法树。

要求测试用例包含函数声明，定义、表达式（各种运算符均在某个表达式中出现）、所有的语句，以及if语句的嵌套，循环语句的嵌套。测试用例中没有出现的语句和嵌套结构，视作没有完成该种语法结构的分析。

常见的语句包括：

1.表达式语句； 2.if语句； 3.if else语句； 4.while语句;

5.for语句； 6. return语句； 7.break语句； 8.continue语句；

9.复合语句； 10.函数定义； 11.函数声明； 12.函数调用；

13. if语句嵌套； 14.循环语句嵌套； 15.外部变量说明语句;

16. 局部变量说明语句。

显示抽象语法树，要求能由抽象语法树说明源程序的语法结构

（2）测试大纲

1.测试目的：通过测试验证系统已经达到设计目标

2.测试环境：

硬件环境：CPU：i7-1165G7，内存：16GB，硬盘：512GB，显卡：2G

软件环境：Windows xp操作系统，Code::Blocks 10.05

3.测试方法

运行程序，输入要测试的文件名。本模块的测试文件名为eg1.c（即没有错误的完整c语言代码）。输入后运行程序。分析结果，检查是否能实现设计目标。

4.测试内容

测试文件：

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #define N 10  int a[100];  int b(int i){ //5  int x;  x=i+1;  return x+056; //Here is a line comment.  }  long n; //10  /\*  \* Here is a block comment.  \*/  void f();  int main(){  int i;  int j;  float x;  x=1.2;  char c;  char s[10];  s[1]='y';  n=12L; //20  c='t';  f();  i=i+j;  for(i=1;i<=n||n>0;){ //Here is a line comment.  a[i+1]=b(i+1)+1;  j\*=i;  for(j=0;j<N;j=j+1)  a[i]=a[i]\*2+1;  }  i=-1;  j=0x1a;  while(j) { //30  i=i<n && (n!=2 || i\*2+j/3-n%4>4);  if(a[i]%2) //32  j=(i+2)\*j/5+6;  else if(a[i]==3) j=j/2-j%(3+i);  else j\*=i;  if(j){ //35  if(j<0) continue;  else j=j-2;  }  else break;  } //40  return 0;  }  void f(){  n=n+1;  return;  } |

以下是对测试文件的分块说明，以证明测试文件包含设计要求中所有功能的测试。

1.基本数据类型的变量、常量，以及数组

|  |  |
| --- | --- |
| 变量 | long n; //10  int i;  float x;  char c; |
| 数组 | int a[100];  char s[10]; |
| 常量 | x=1.2;  s[1]='y';  n=12L; //20  c='t';  i=-1;  j=0x1a;  return x+056; //Here is a line comment. |

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

|  |
| --- |
| a[i+1]=b(i+1)+1;  j\*=i;  i=i<n && (n!=2 || i\*2+j/3-n%4>4); //31 |

3.函数定义、声明与调用

|  |  |
| --- | --- |
| 函数定义 | int b(int i){ //5  int x;  x=i+1;  return x+056; //Here is a line comment.  } |
| 函数声明 | void f(); |
| 函数调用 | f();  a[i+1]=b(i+1)+1; |

4.所有语句

|  |  |
| --- | --- |
| 表达式语句 | j=(i+2)\*j/5+6; |
| 复合语句 | void f(){  n=n+1;  return;  } |
| if-else语句 | if(j<0) continue;  else j=j-2; |
| if-else-if语句 | if(a[i]%2)  j=(i+2)\*j/5+6;  else if(a[i]==3) j=j/2-j%(3+i);  else j\*=i; |
| while语句 | while(j) { //30  i=i<n && (n!=2 || i\*2+j/3-n%4>4);  if(a[i]%2) //32  j=(i+2)\*j/5+6;  else if(a[i]==3) j=j/2-j%(3+i);  else j\*=i;  if(j){ //35  if(j<0) continue;  else j=j-2;  }  else break;  } //40 |
| for语句 | for(j=0;j<N;j=j+1)  a[i]=a[i]\*2+1; |
| return语句 | return x+056; //Here is a line comment.  return 0; |
| break语句 | else break; |
| continue语句 | if(j<0) continue; |
| 外部变量说明 | long n; |
| 局部变量说明 | int i; |
| if语句嵌套 | if(j){ //35  if(j<0) continue;  else j=j-2;  }  else break; |
| for语句嵌套 | for(i=1;i<=n||n>0;){ //Here is a line comment.  a[i+1]=b(i+1)+1;  j\*=i;  for(j=0;j<N;j=j+1)  a[i]=a[i]\*2+1;  } |

5.编译预处理

|  |  |
| --- | --- |
| 头文件包含 | #include<stdio.h>  #include<stdlib.h> |
| 宏定义 | #define N 10 |

6.注释

|  |  |
| --- | --- |
| 行注释 | for(i=1;i<=n||n>0;){ //Here is a line comment. |
| 块注释 | /\*  \* Here is a block comment.  \*/ |

（3）运行结果

本模块的测试结果需要通过输出AST模块来可视化。

运行截图：

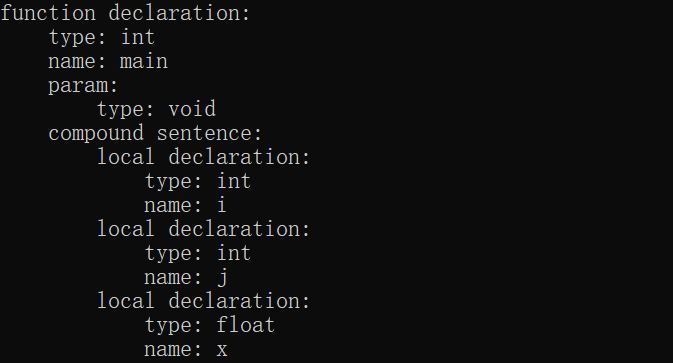
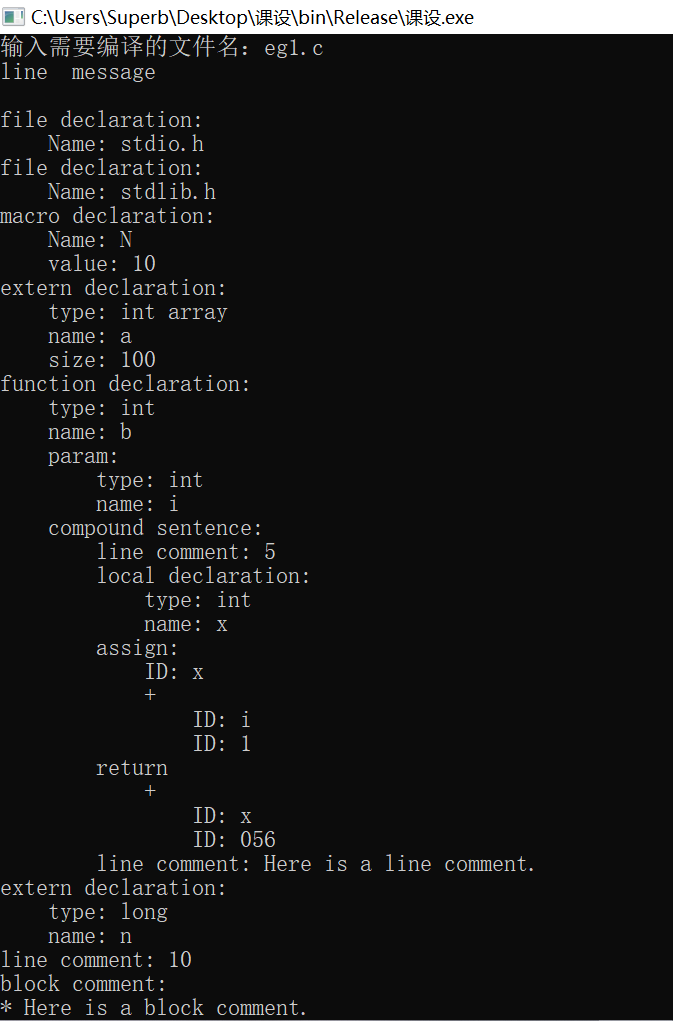


图4-1 运行截图1

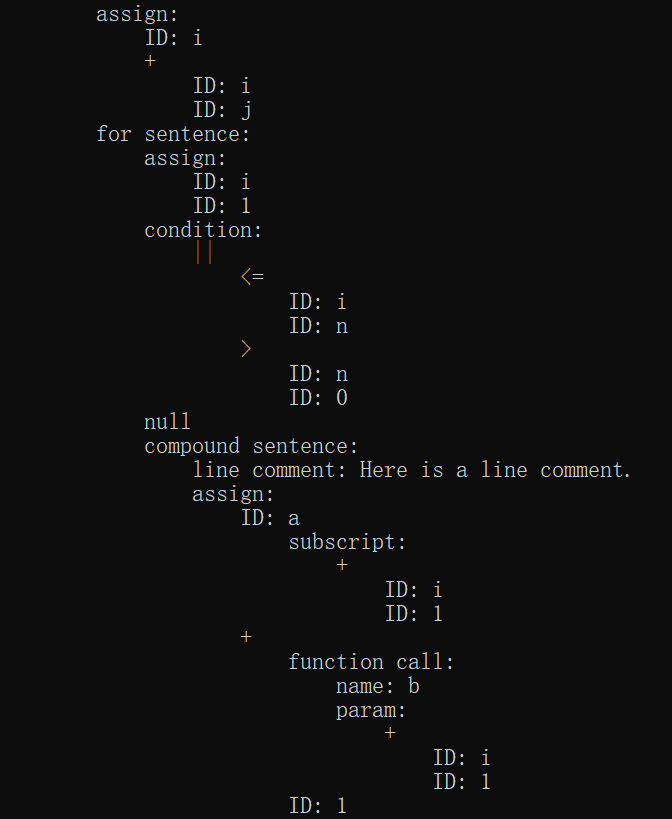
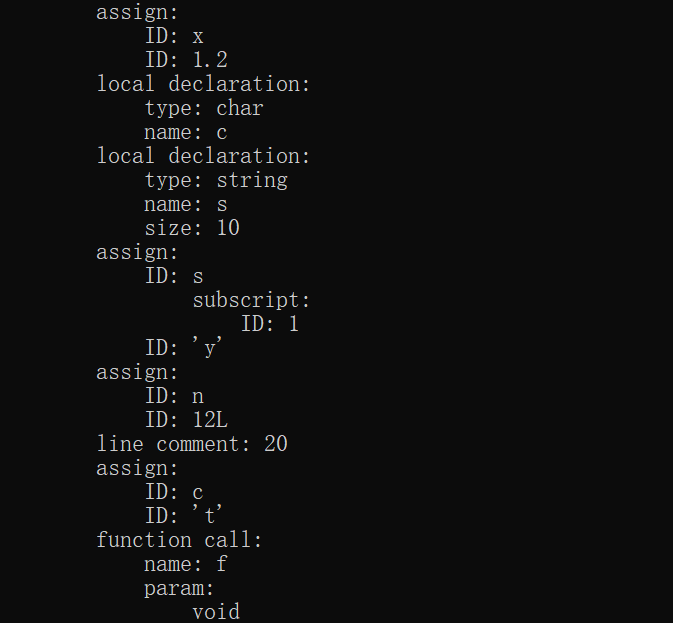


图4-2 运行截图2

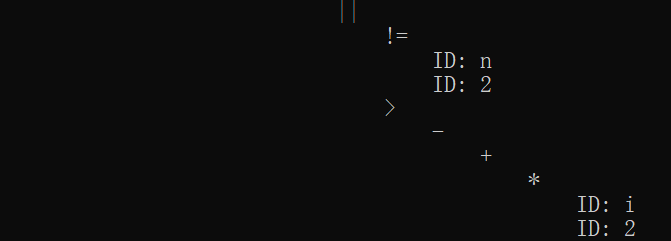
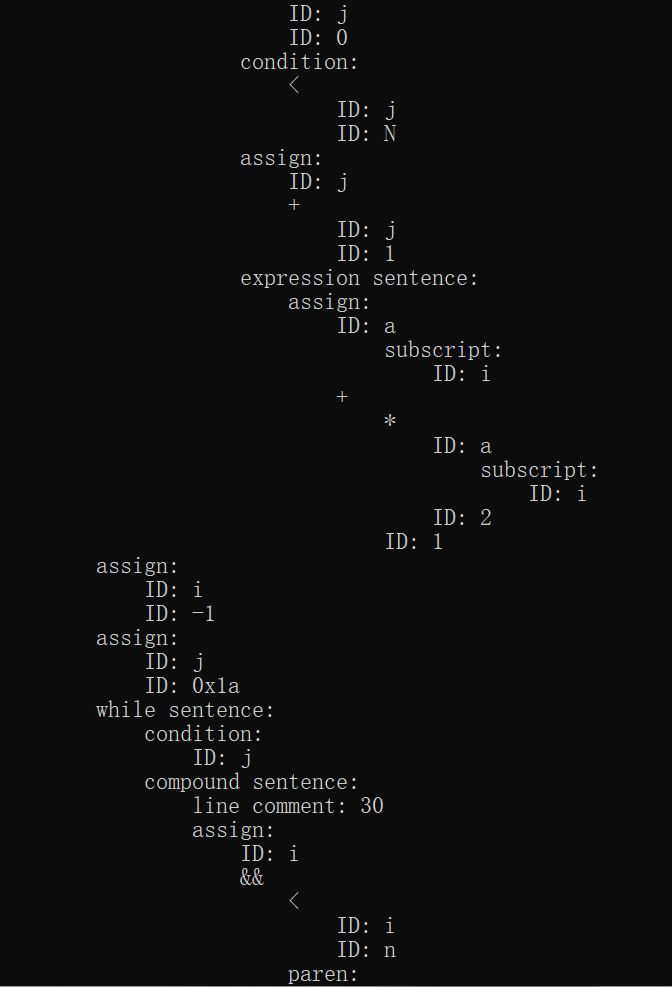
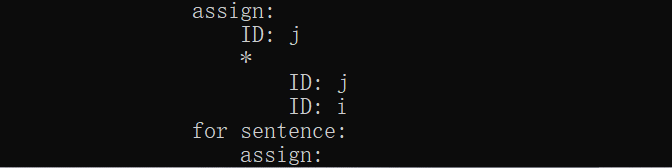


图4-3 运行截图3

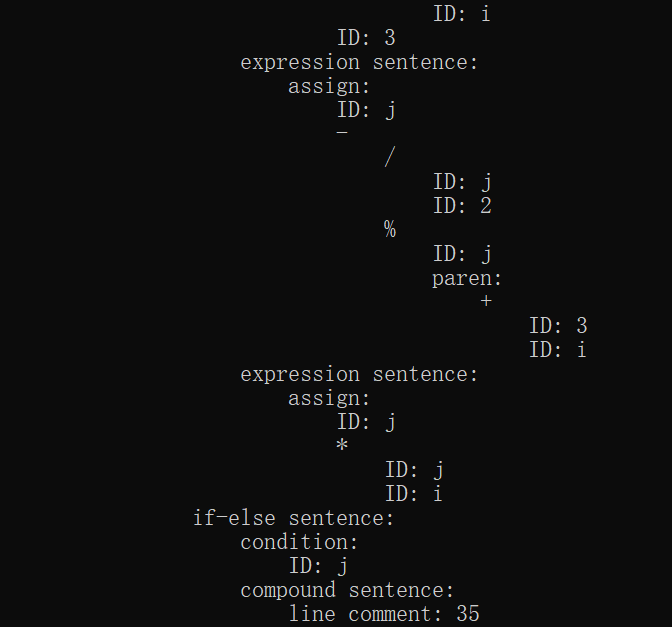
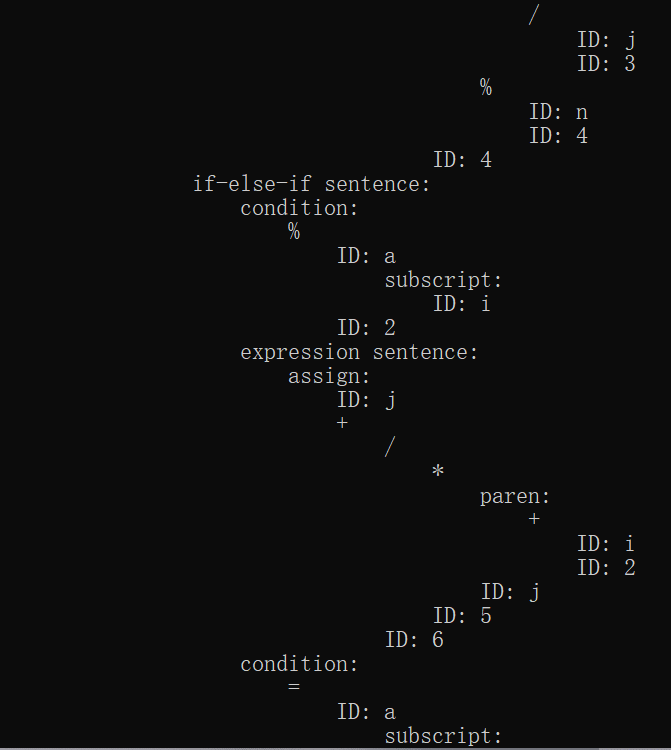


图4-4 运行截图4

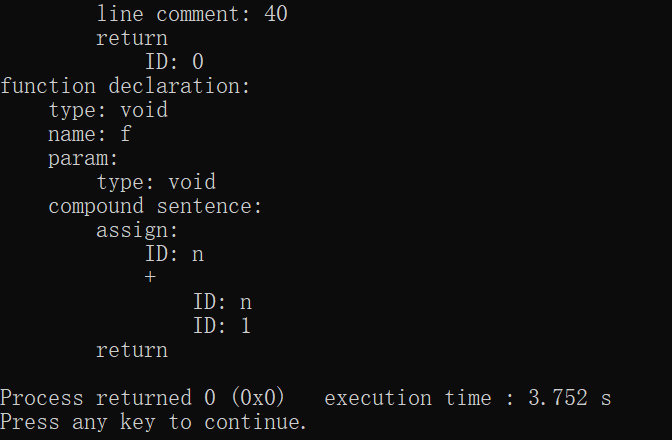
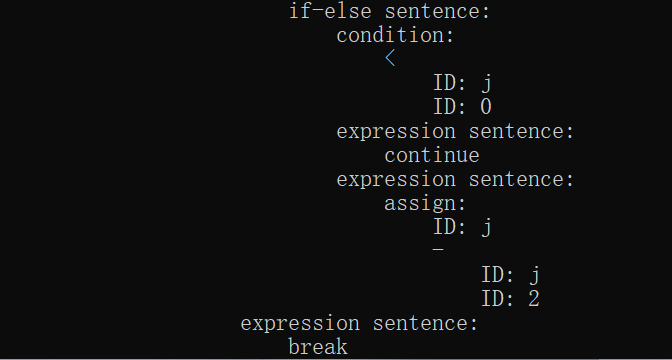


图4-5 运行截图5

（4）分析

该模块能够处理没有错误的完整c语言代码并生成AST，以下是说明：

1.变量、常量、数组、外部及局部变量定义



图4-6 外部变量定义&数组

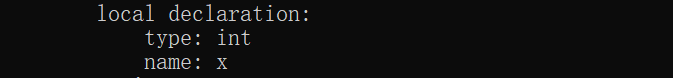


图4-7 局部变量定义



图4-8 变量&常量

2.所有运算符

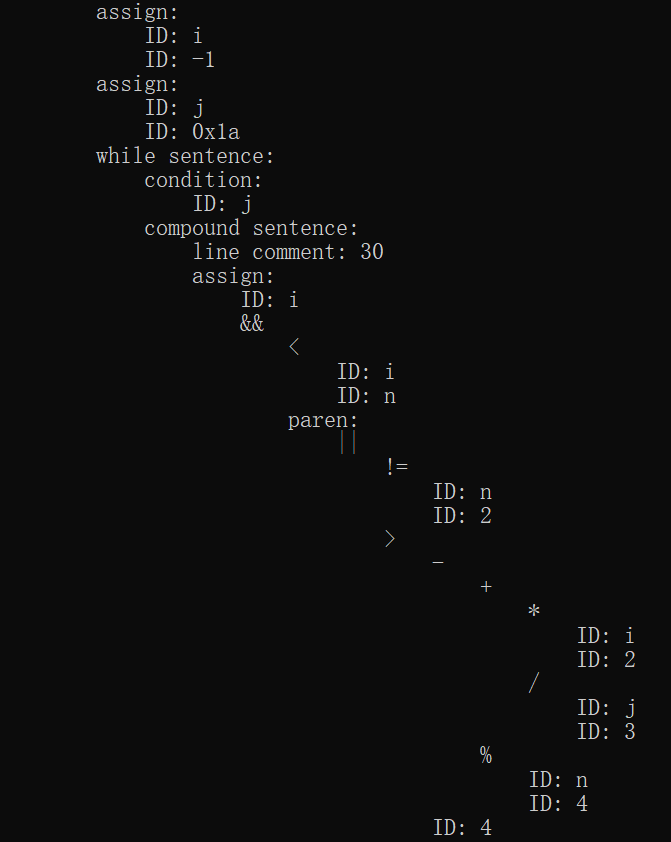


图4-9 运算符

3.函数定义、声明与调用

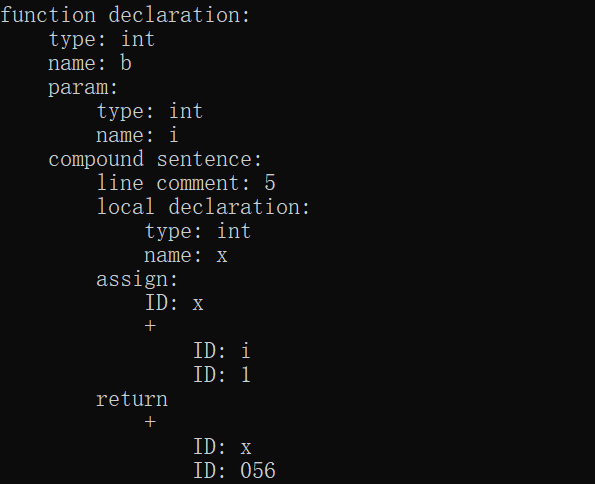


图4-10 函数定义

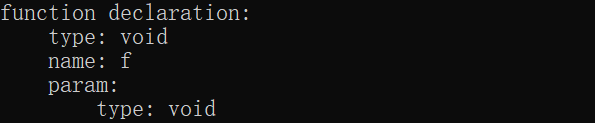


图4-11 函数声明

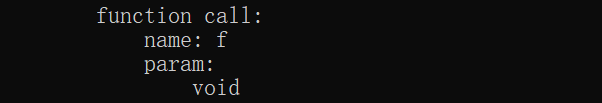


图4-12 函数调用

4.所有语句

1)for语句

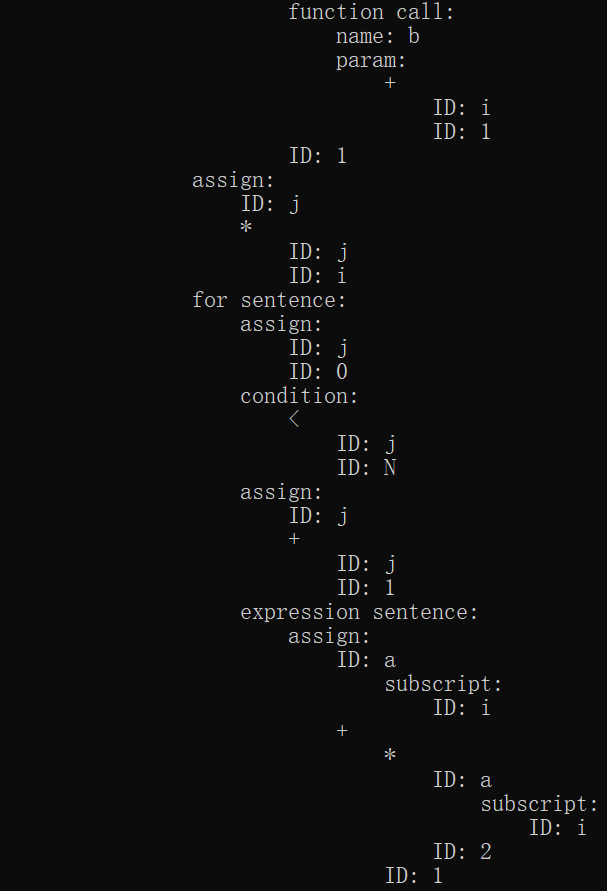


图4-13 for语句

2)if语句

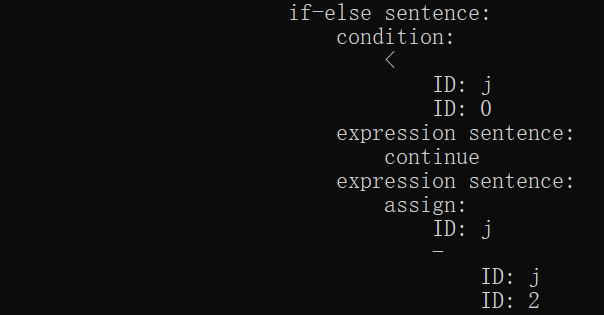


图4-14 if-else语句

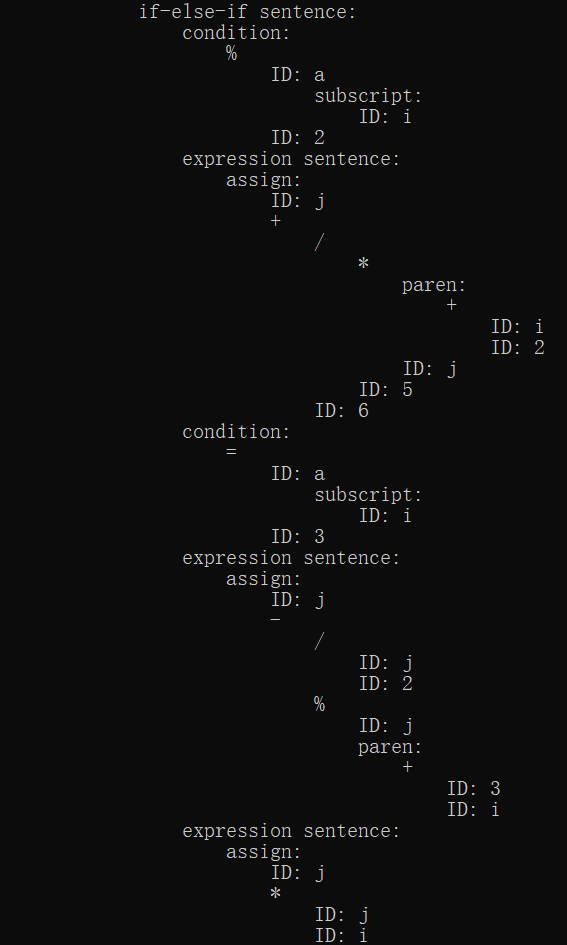


图4-15 if-else-if语句

3)while语句

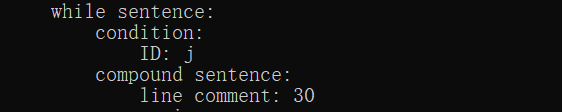


图4-16 while语句

4)break语句



图4-17 break语句

5)continue语句



图4-18 continue语句

6)return语句

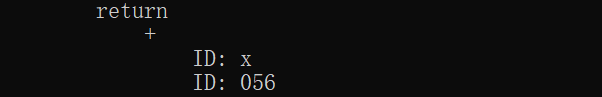


图4-19 return语句

7)复合语句

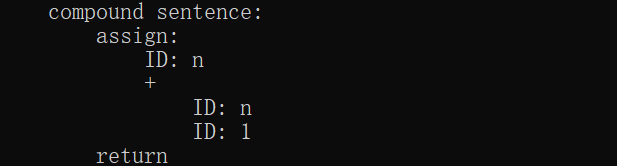


图4-20 复合语句

8)循环语句嵌套

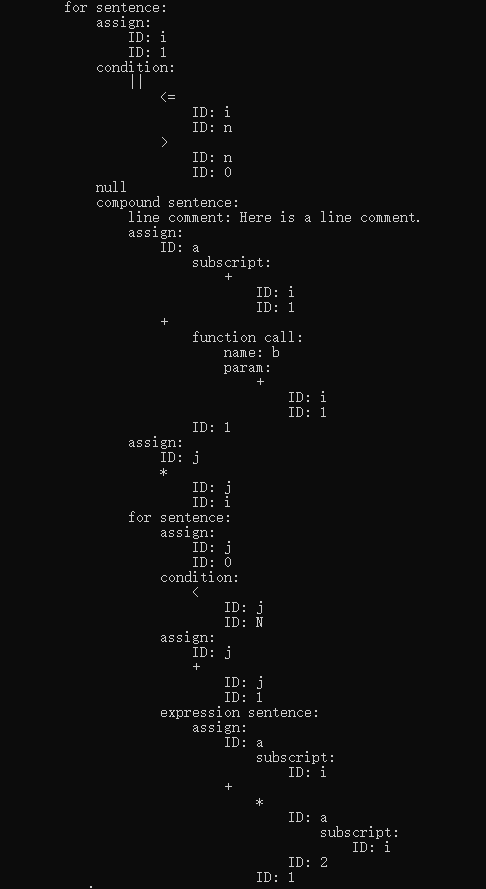


图4-21 循环语句嵌套

9)if语句嵌套

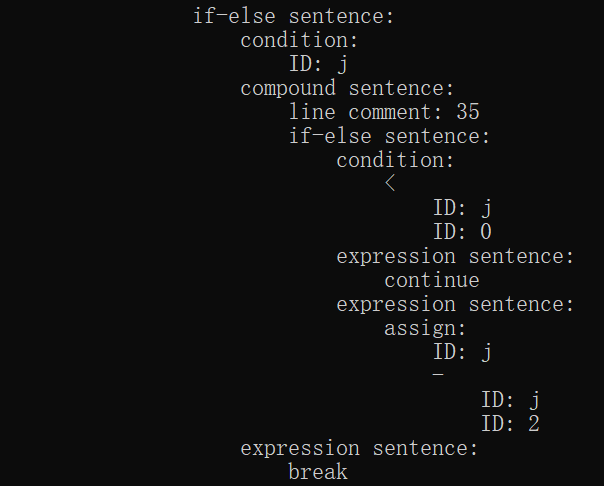


图4-22 if语句嵌套

5.编译预处理

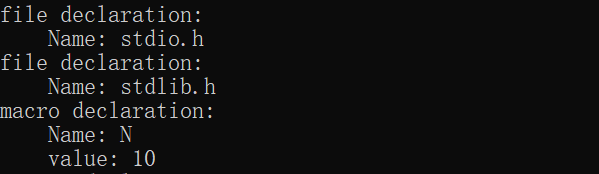


图4-23 编译预处理

6.注释

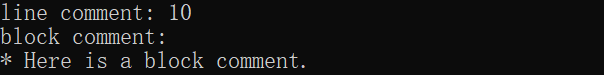


图4-24 注释

2.报错模块

（1）功能与设计目标

报错功能，指出不符合语法规则的错误位置和不符合单词定义的符号位置。

（2）测试大纲

1.测试目的：通过测试验证系统已经达到设计目标

2.测试环境：

硬件环境：CPU：i7-1165G7，内存：16GB，硬盘：512GB，显卡：2G

软件环境：Windows xp操作系统，Code::Blocks 10.05

3.测试方法

运行程序，输入要测试的文件名。本模块的测试文件名为eg2.c（有错误语法和错误符号的完整c语言代码）。输入后运行程序。分析结果，检查是否能实现设计目标。

4.测试内容

测试文件：

|  |
| --- |
| #include stdio.h>  #include "stdlib.h"  #define N1 10  int a[100];  int b(int i){ //5  int x;  x=i+1  return x+058; //Here is a line comment.  }  long n; //10  /\*  \* Here is a block comment.  \*/  void f(); @  int main(){  int i;  int j;  long i;  float 0x;  x=1.2.3;  char c;  char s[10];  s[1]='yz';  n=12Ll; //20  c[0]='t';  f(1);  i=i+j;  for(i=1;i<=n||n>0; { //Here is a line comment.  a[i+1]=b()+1;  j\*=i;  for(j=0;j<n;j+1)  a=a[i]\*2+1;  }  i=-1;  j=0x1a;  break;  while j) { //30  i=i<n && (n!=2 || i\*2+j/3-n%4>4); //31  if(a[i%2) //32  j=(i+2)\*j/5+6;  else if(a[i]==3) j=j/2-j%(3+i); //33  else j\*=i;  if(){ //35  if(j<0) continue;  else j=j-2;  else j=2;  }  else break;  } //40  return;  }  void f(){  n=n+1;  return 0; |

以下对文件中错误的语句和符号部分进行说明

|  |  |
| --- | --- |
| 错误符号 | @ |
| 头文件界定符错误 | #include stdio.h> |
| 宏定义命名错误 | #define N1 10 |
| 标识符错误 | return x+058; //Here is a line comment. |
| 变量名重复 | int i;  long i; |
| 变量命名错误 | float 0x; |
| 数字错误 | x=1.2.3; |
| 变量未定义 | x=1.2.3; |
| 赋值字符过长 | s[1]='yz'; |
| 数字错误 | n=12Ll; //20 |
| 变量类型错误 | c[0]='t';  a=a[i]\*2+1; |
| 函数参数错误 | f(1);  a[i+1]=b()+1; |
| 分隔符号缺失 | x=i+1 |
| 括号缺失 | for(i=1;i<=n||n>0; { //Here is a line comment.  while j)  if(a[i%2) //32  void f(){  n=n+1;  return 0; |
| 无效果语句 | for(j=0;j<n;j+1) |
| break，continue语句不在循环语句中 | break; |
| 条件缺失 | if(){ //35 |
| else缺少if | else j=2; |
| 函数返回类型错误 | return;  return 0; |

（3）运行结果

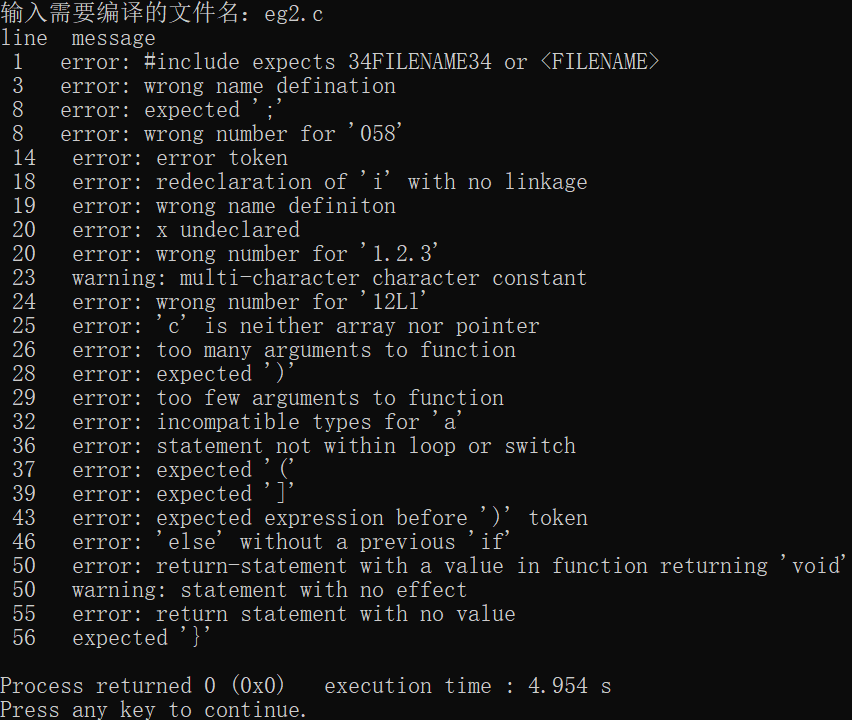


图4-25 运行截图

（4）分析

该模块能够完成表格中的所有报错功能。

以下表格将错误内容和报错内容进行对应。

|  |  |
| --- | --- |
| 错误符号 | @ |
|  | |
| 头文件界定符错误 | #include stdio.h> |
|  | |
| 宏定义命名错误 | #define N1 10 |
|  | |
| 标识符错误 | return x+058; //Here is a line comment. |
|  | |
| 变量名重复 | int i;  long i; |
|  | |
| 变量命名错误 | float 0x; |
|  | |
| 数字错误 | x=1.2.3; |
|  | |
| 变量未定义 | x=1.2.3; |
|  | |
| 赋值字符过长 | s[1]='yz'; |
|  | |
| 数字错误 | n=12Ll; //20 |
|  | |
| 变量类型错误 | c[0]='t';  a=a[i]\*2+1; |
|  | |
| 函数参数错误 | f(1);  a[i+1]=b()+1; |
|  | |
| 分隔符号缺失 | x=i+1 |
|  | |
| 括号缺失 | for(i=1;i<=n||n>0; { //Here is a line comment.  while j)  if(a[i%2) //32  void f(){  n=n+1;  return 0; |
|  | |
| 无效果语句 | for(j=0;j<n;j+1) |
|  | |
| break，continue语句不在循环语句中 | break; |
|  | |
| 条件缺失 | if(){ //35 |
|  | |
| else缺少if | else j=2; |
|  | |
| 函数返回类型错误 | return;  return 0; |
|  | |

3.源文件编排模块

（1）功能与设计目标

缩进编排重新生成源程序文件。对测试模块1生成的抽象语法树进行遍历，按缩进编排的方式写到.c文件中，查看文件验证是否满足任务要求。

（2）测试数据

1.测试目的：通过测试验证系统已经达到设计目标

2.测试环境：

硬件环境：CPU：i7-1165G7，内存：16GB，硬盘：512GB，显卡：2G

软件环境：Windows xp操作系统，Code::Blocks 10.05

3.测试方法

运行程序，输入要测试的文件名。本模块的测试文件名为eg3.c（没有错误的缩进错误的代码）。输入后运行程序。分析结果，检查是否能实现设计目标。

4.测试内容

测试文件：

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #define N 10  int a[100];  int b(int i){ //5  int x;  x=i+1;  return x+056; //Here is a line comment.  }  long n; //10  /\*  \* Here is a block comment.  \*/  void f();  int main(){  int i;int j;  float x; x=1.2;  char c;  char s[10];  s[1]='y';  n=12L; //20  c='t';f();  i=i+j;  for(i=1;i<=n||n>0;){ //Here is a line comment.  a[i+1]=b(i+1)+1;  j\*=i;  for(j=0;j<N;j=j+1)  a[i]=a[i]\*2+1;  }i=-1;  j=0x1a;  while(j) { //30  i=i<n && (n!=2 || i\*2+j/3-n%4>4);  if(a[i]%2)  j=(i+2)\*j/5+6;  else if(a[i]==3) j=j/2-j%(3+i);  else j\*=i;  if(j){ //35  if(j<0) continue;  else j=j-2;  }  else break;  } //40  return 0;  }  void f(){n=n+1;return;} |

（3）运行结果

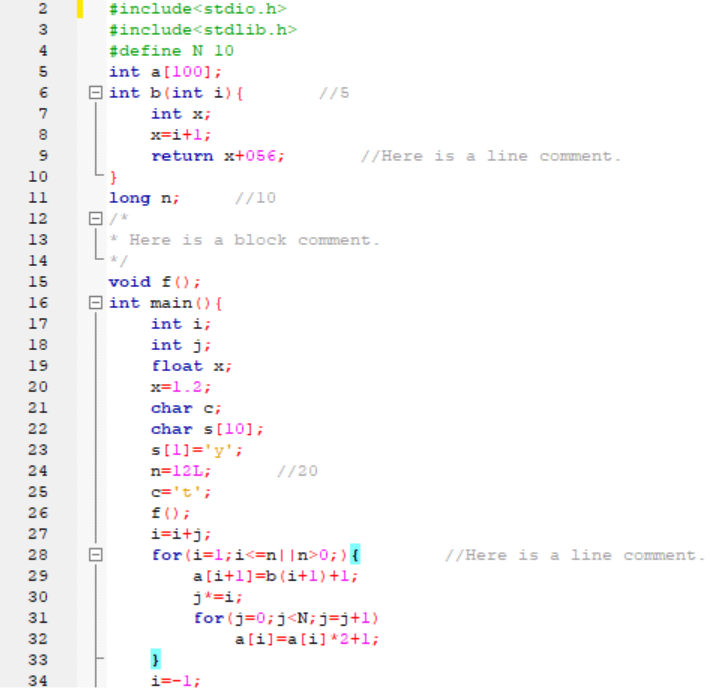


图4-26 运行截图1

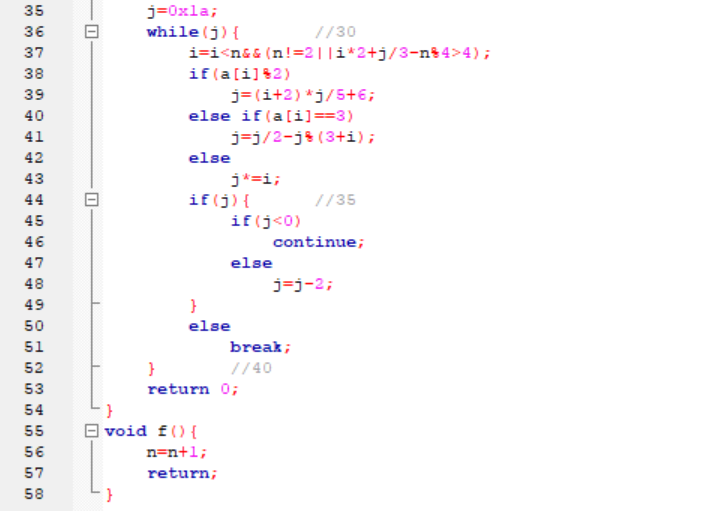


图4-27 运行截图2

（4）分析

该模块能够编排缩进，将AST中的内容用c语言代码表示出来。

# 5 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）查阅资料，了解抽象语法树的功能、形式和生成方法

（2）根据设计要求设计编写符合要求的测试文件

（3）测试编写生成AST的程序

（4）对测试用例进行测试，根据测试结果修改程序，使其功能符合设计要求。

（5）写实验报告

## 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）程序运行效率的提高

（2）对更多种类语句（switch、do-while等）、运算符（逗号运算符、位运算符、单目运算符等）的处理、各种头文件包含的函数（scanf、printf、strcpy等）的处理

（3）程序稳定性的提升，不会因为读到语法错误而中断运行。

# 6 体会

在开始编写测试文件和代码之前，我在查找参考资料时遇到了很大问题，因为网上的大部分关于AST的资料都基于java script语言，而我对这种语言没有任何了解。因此我只能对AST的概念和原理进行了解，并尝试使用了几个在线生成AST网站。但任务书要求的功能不多，我能够运用原来掌握的知识来实现。在写代码和调试过程中，我对函数的调用、递归以及树的各种功能的实现有了更熟练的掌握。

**参考文献**

[1]https://en.wikipedia.org/wiki/Abstract\_syntax\_tree

[2]https://www.cnblogs.com/qinmengjiao123-123/p/8648488.html

[3]<https://blog.csdn.net/weixin_39408343/article/details/95984062>

[4]<https://zhuanlan.zhihu.com/p/102385477>

[5]<https://juejin.cn/post/6844903725228621832>

[6]https://zhuanlan.zhihu.com/p/81877656

[7]https://github.com/jamiebuilds/the-super-tiny-compiler