
北京邮电大学

计算机学院

# C 语言词法分析程序的设计与实现

(C++ 版和 FLEX 版均实现)

学号: 2020211597

班级: 2020211323

指导老师: 王雅文

课程名称: 编译原理

October 3, 2022

# Contents

# 1 题目及要求

1. 设计一个 C 语言词法分析程序;
2. 可以识别出用 C 语言编写的源程序中的每个单词符号, 运算符, 数字, 字符 (包括转义字符) 等, 并以 < 记号, 属性 > 的形式输出每个单词符号;
3. 可以识别并跳过源程序中的注释;
4. 可以统计源程序中的语句行数, 各类单词个数, 字符总数, 并输出统计结果;
5. 可以识别 C 语言程序源代码中存在的词法错误, 并报告错误出现的位置;
6. 对源程序中出现的错误进行适当的恢复, 让词法分析可以继续进行;
7. 对源程序进行一次扫描, 即可检查并报告源程序中存在的所有词法错误, 并输出源程序中所有的记号.

# 2 实验设备

**操作系统** Ubuntu 20.04.5 LTS on Windows 10 x86_64,
macOS 12.6 21G115 arm64,

**文本编辑器** Neovim v0.7.2,

**编译器** clang-1400.0.29.102,

**FLEX** flex 2.6.4

# 3 程序设计说明

1. 词法分析程序的作用:
   (a) 扫描源程序字符流;
   (b) 按照源语言的词法规则识别出各类单词符号;
   (c) 产生用于语法分析的记号序列;
   (d) 词法检查;
   (e) 创建符号表, 将识别出来的标识符放入符号表中;
   (f) 跳过源程序中的注释和空格等, 把错误信息和源代码联系起来;
2. 源程序代词类别:
   (a) 关键字;
   (b) 用户定义变量标识符;
   (c) 数字, 字符和字符串常量;
   (d) 运算符;

(e) 分隔符;

3. 设计思路: 利用有限状态自动机模型, 将整个源代码分析的过程转化为不同状态之间的转移, 在画好状态转移图之后, 借用 C++ 的 switch 语句或 if/else 语句将状态转移图描述出来. 此外, 实现好读取源代码, 缓冲区, 以及输出分析结果, 和将常量和变量名插入到符号表的功能.

4. 伪代码描述:

```
// Initializing...
while(End of source file not reached) {
  ch = getchar();
  switch(ch) {
    determine state based on the input char
    case 0:
    ...
  }
  switch(State) {
    // Limited state machine process
    case 0:
      // state 0 process
    case 1:
      // state 1 process
    ...
  }
}
```

# 4 实验流程图

用状态转移图描述的词法分析程序如下图 1和图 2.
其中状态 0 是起始状态, 状态 EXIT 为当源代码文件未分析完毕时回到状态 0, 分析完毕时退出.

# 5 实验程序

## 5.1 C++ 版实现

出于模块化, 解耦合, 可扩展性和可读性考虑, 将词法分析程序划分为符号和符号表与词法分析处理和输入输出两个模块, 分别在 Symbol.h, Symbol.cpp, Lex.h 和 Lex.cpp 四个源代码文件中进行实现, 并使用 CMake 作为构建工具.

其中, Symbol.h 和 Symbol.cpp 对记号 (Symbol) 及记号表 (SymbolList) 的类进行成员和方法定义和实现, Lex.h 和 Lex.cpp 对词法分析处理, 输入和输出类 Lex 进行成员和方法定义和实现.

main.cpp 承担着作为词法分析程序入口的功能.

### 5.1.1 main.cpp

词法分析程序入口

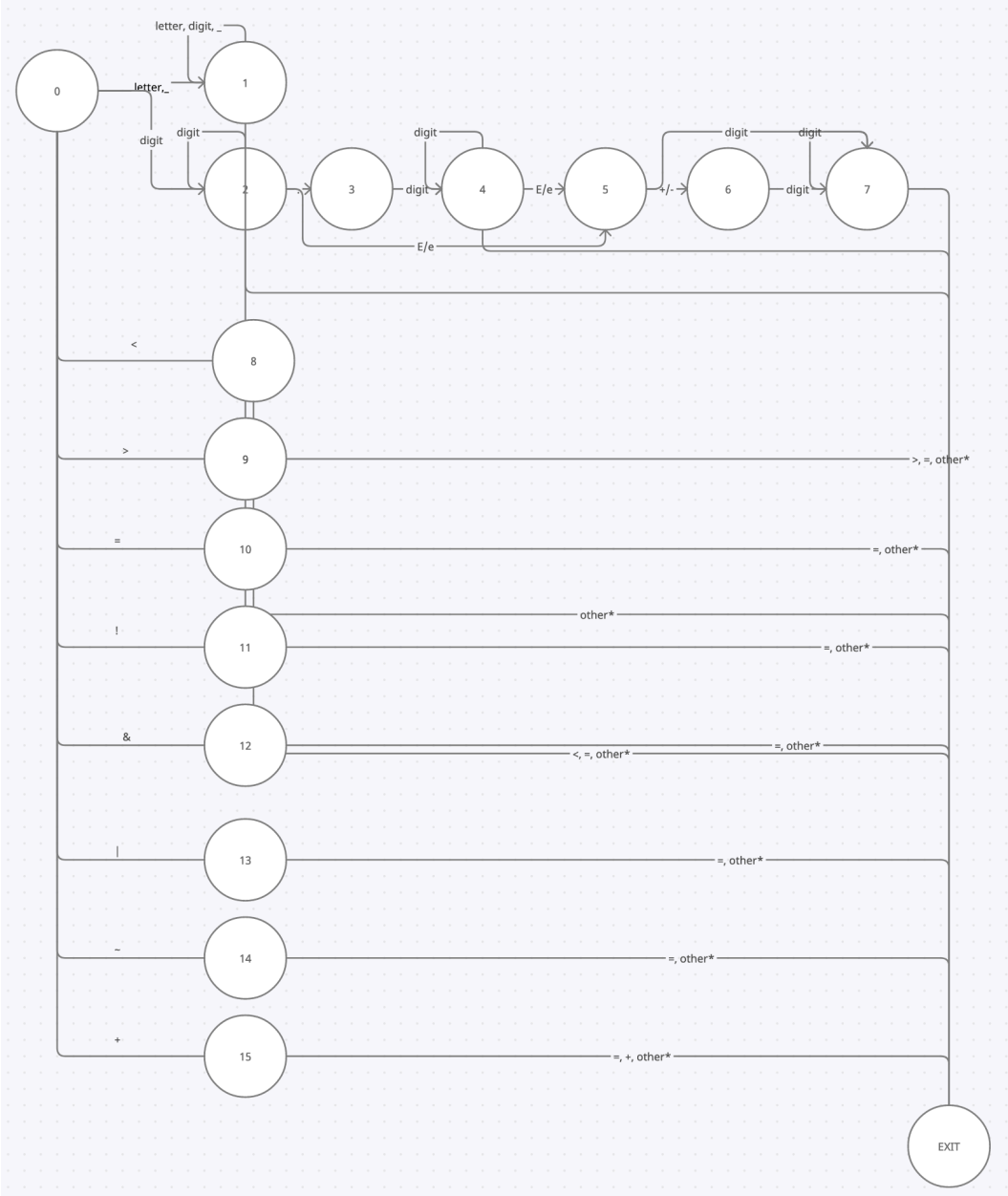


Figure 1: 词法分析程序的状态转移图 1

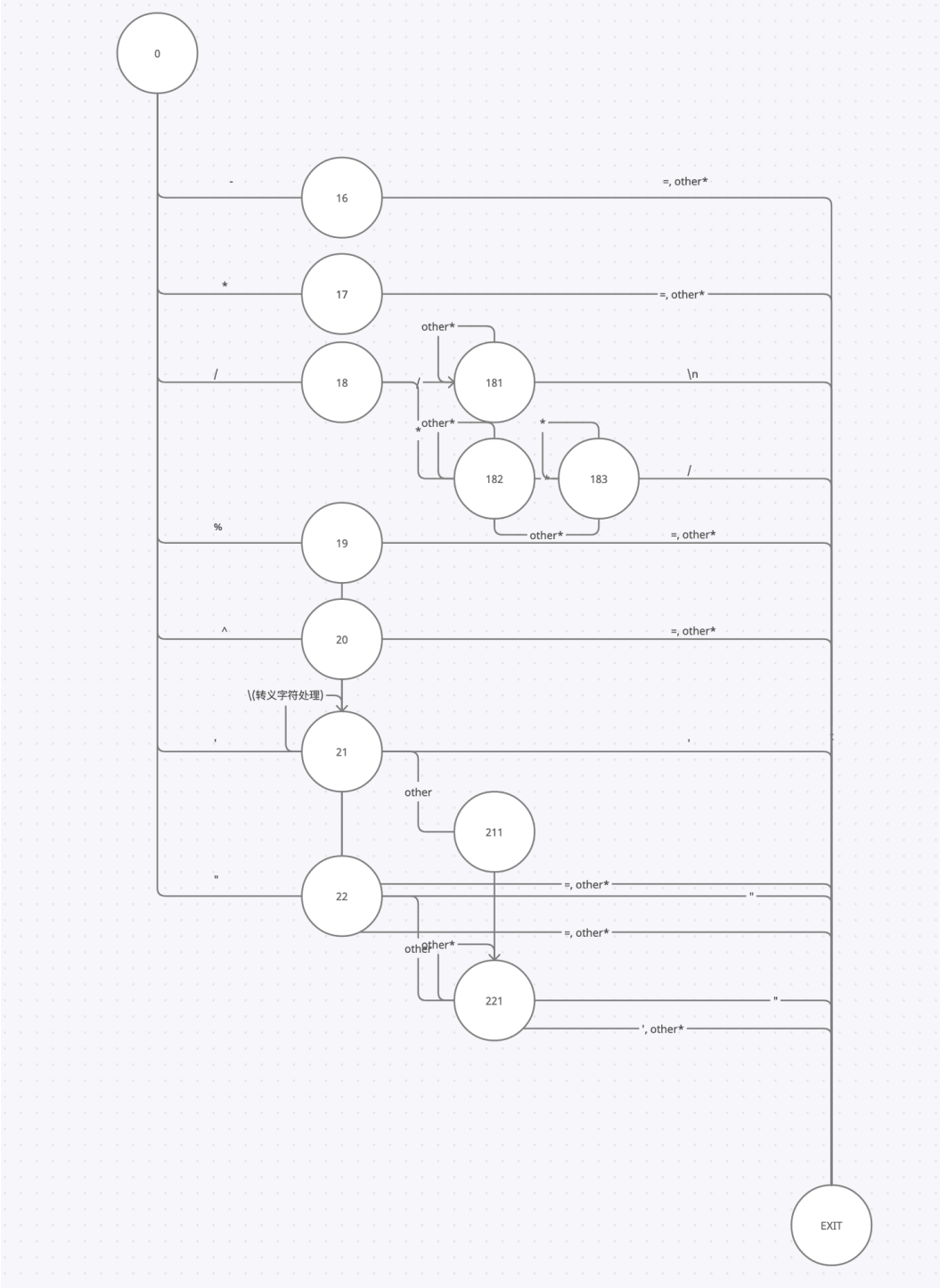


Figure 2: 词法分析程序的状态转移图 2

```
#include "Lex.h"
#include "Symbol.h"

using namespace parser;
using namespace std;

int main(int argc, char **argv) {
    ifstream fs;
    // 如果命令行参数为空，则从默认输入文件sin读入待分析源代码
    // 否则读入命令行参数给出的文件作为待分析源代码
    string fName = FNAME;
    if(argc > 1) {
        fName = argv[1];
    }
    Lex lex(fName);
    lex.process();
    lex.print();

    return 0;
}
```

### 5.1.2 Symbol.h

对记号 (Symbol) 及记号表 (SymbolList) 的类成员和方法定义;

```
#ifndef SYMBOL_H
#define SYMBOL_H

#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_set>
#include <vector>

namespace parser {
using namespace std;
class Symbol {
  public:
    Symbol() : notation("NULL"), property("NULL"), count(1) {}
    Symbol(string _notation, string _property, int _count = 1)
        : notation(_notation), property(_property), count(_count) {}
    string getNotation() { return notation; }
    string getProperty() { return property; }
    void incCount() { count++; }
    int getCount() { return count; }
    string toString() {
      return "<" + notation + ", " + property + ">";
    }
    string toStringWithCount() {
        return "<" + notation + ", " + property + ">"
          + " appeared " + to_string(getCount()) + " times";
    }
    bool operator==(const Symbol &rSymbol) {
        // Two symbols have same notation and property
        return
```

```
            (notation == rSymbol.notation && property == rSymbol.property);
    }
    friend ostream &operator<<(ostream &os, Symbol &symbol);

  private:
    // symbol' notation
    string notation;
    // symbol's property
    string property;
    // count of symbol appearance time
    int count;
};

class SymbolList {
  public:
    SymbolList() : numSymbol(0) {}
    void add(Symbol &symbol);
    void
    add(const string &notation,
        const string &property);
    vector<Symbol>::iterator find(Symbol &symbol);
    vector<Symbol>::iterator find(const string &notation,
                                  const string &property);
    int getNumSymbol() const { return numSymbol; }
    Symbol find(const string &pattern);
    string toString();
    void print();
    Symbol &operator[](const int &i);
    friend ostream &operator<<(ostream &os, SymbolList &symbolList);

  private:
    vector<Symbol> sl;
    int numSymbol;
};
} // namespace parser
#endif
```

### 5.1.3 Symbol.cpp

对记号 (Symbol) 及记号表 (SymbolList) 的类成员和方法实现;

```
#include "Symbol.h"

namespace parser {
using namespace std;

// class Symbol
ostream &operator<<(ostream &os, Symbol &symbol) {
    os << symbol.toStringWithCount();
    return os;
}

// class SymbolList
vector<Symbol>::iterator SymbolList::find(Symbol &symbol) {
    for (auto iter = sl.begin(); iter != sl.end(); iter++) {
        if (symbol == *iter) {
            return iter;
```

```
        }
    }
    return sl.end();
}

vector<Symbol>::iterator SymbolList::find(const string &notation,
                                          const string &property) {
    Symbol sym(notation, property);
    return find(sym);
}

void SymbolList::add(Symbol &symbol) {
    auto idxSymbol = find(symbol);
    if (idxSymbol == sl.end()) {
        // Symbol that has never appeared before
        sl.push_back(Symbol(symbol));
        numSymbol++;
    } else {
        // Symbol that has already exists in symbolList
        idxSymbol->incCount();
    }
}

void SymbolList::add(const string &notation, const string &property) {
    Symbol sym(notation, property);
    add(sym);
}

/**
 * find symbol in SymbolList by pattern(notation or property)
 */
Symbol SymbolList::find(const string &pattern) {
    for (auto iter = sl.begin(); iter != sl.end(); iter++) {
        if (iter->getNotation() == pattern ||
                iter->getProperty() == pattern) {
            return *iter;
        }
    }
    return Symbol();
}

Symbol &SymbolList::operator[](const int &i) { return sl[i]; }

ostream &operator<<(ostream &os, SymbolList &symbolList) {
    int numSymbol = symbolList.getNumSymbol();
    os << numSymbol << "␣symbols" << endl;
    for (int i = 0; i < numSymbol; i++) {
        os << symbolList[i] << endl;
    }
    return os;
}

/**
 * output SymbolList as string
 */
string SymbolList::toString() {
    stringstream ss;
    ss << (*this);
```

```
    return ss.str();
}

/**
 * print SymbolList to stdout
 */
void SymbolList::print() { cout << this->toString(); }
} // namespace parser
```

### 5.1.4 Lex.h

词法分析处理和输出类 Lex 的成员和方法定义;

```
#ifndef PARSER_H
#define PARSER_H

#include "Symbol.h"
#include "Util.h"

#include <algorithm>
#include <fstream>
#include <iostream>
#include <string>
#include <unordered_set>
#include <vector>

const int BUFFER_SIZE = 1024;
const std::string FNAME = "sin";

namespace parser {
using namespace std;

class Lex {
  public:
    Lex(const string &filename);

    ~Lex() {
        try {
            fs.close();
        } catch (ifstream::failure &e) {
            cout << "Exception␣closing␣file.\n";
        }
    }
    void process();
    void print();

  private:
    unordered_set<string> keywords;
    ifstream fs;
    SymbolList sl;
    char buffer[BUFFER_SIZE];
    int numLines;
    int numChar;
    char ch;        // store next char
    string bufStr; // store current string
    int pForward;  // forward pointer of buffer
    int pBackward;
```

```
    void get_char() {
        pForward = (pForward + 1) % BUFFER_SIZE;
        ch = buffer[pForward];
    }
    bool is_letter() { return (isalpha(ch) || ch == '_'); }
    bool is_digit() { return (ch >= '0' && ch <= '9'); }
    bool is_keyword() {
      return (keywords.find(bufStr) != keywords.end());
    }
    void cat() {
        bufStr.push_back(ch);
    } // concat ch to the end of bufStr.
    void clrStr() { bufStr.clear(); }
    void unget_char() {
        pForward = (pForward - 1) % BUFFER_SIZE;
        // decrease char counter when
        // ch is not EOF
        if (ch != EOF) {
            numChar--;
        }
        // decrease line counter when
        // ch is end of line symbol '\n'
        if (ch == '\n') {
            numLines--;
        }
    }

    void addSymbol(const string &notation, const string &property = "") {
        Symbol sym(notation, property);
        sl.add(sym);
        cout << "L" << numLines+1 << ": " << sym.toString() << endl;
    }

    // log zone
    void logError(const string &err) {
        string msg = "ERROR: " + err + "in L" + to_string(numLines+1);
        cout << msg << endl;
    }
};
} // namespace parser
#endif
```

### 5.1.5 Lex.cpp

词法分析处理和输出类 Lex 的成员和方法实现;

```
#include "Lex.h"

namespace parser {
using namespace std;

// Lex
Lex::Lex(const string &filename) : numLines(0), numChar(0), pForward(-1) {
    // load c source file
    try {
        fs.open(filename);
```

```
    } catch (ifstream::failure &e) {
        cout << "Exception␣opening␣file.\n";
    }
    vector<string> vecKeywords(
        {"auto",   "switch",   "case",     "for",    "do",     "while",
         "int",    "char",     "float",    "double", "if",     "else",
         "return", "break",    "continue", "const",  "enum",   "extern",
         "goto",   "register", "restrict", "short",  "signed", "unsigned",
         "sizeof", "static",   "inline",   "struct", "class",  "typedef",
         "union",  "void",     "volatile"});
    keywords = unordered_set<string>(vecKeywords.begin(), vecKeywords.end());
};

void Lex::process() {
    // end of analysis process sign
    bool end = false;
    // alias to fs
    ifstream &f = this->fs;
    try {
        f.read(buffer, BUFFER_SIZE - 1);
    } catch (ifstream::failure &e) {
        cout << "Read␣to␣buffer␣failure.\n";
    }
    if (f.gcount() < BUFFER_SIZE - 1) {
        buffer[f.gcount()] = EOF;
    }
    buffer[BUFFER_SIZE - 1] = EOF;

    // initial state
    int state = 0;
    while (!end) {
        // cout << "pForward[" << pForward << "]" << endl;
        get_char();
        // cout << "pForward[" << pForward << "]" << endl;
        // cout << "ch[" << ch << "]" << endl;
        if (ch != EOF) {
            numChar++;
        }
        if (ch == EOF && pForward == BUFFER_SIZE - 1) {
            f.read(buffer, BUFFER_SIZE - 1);
            if (f.gcount() < BUFFER_SIZE - 1) {
                buffer[f.gcount()] = EOF;
            }
            continue;
        }
        // EOF in source file, should not be transparent to Lex
        if (ch == EOF && pForward != BUFFER_SIZE - 1) {
            end = true;
        }
        if (ch == '\n') {
            numLines++;
        }

        switch (state) {
        case 0:
            if (is_letter()) {
                // letter state
                state = 1;
```

```
            cat();
        } else if (is_digit()) {
            if (ch != '0') {
                // integer state
                state = 2;
                cat();
            } else if (ch == '0') {
                // TODO: implement hex, bin and octtal decimal mode.
                // state =
                // cat();
            }
        } else {
            switch (ch) {
            case '<':
                state = 8;
                break;
            case '>':
                state = 9;
                break;
            case '=':
                state = 10;
                break;
            case '!':
                state = 11;
                break;
            case '&':
                state = 12;
                break;
            case '|':
                state = 13;
                break;
            case '~':
                state = 14;
                break;
            case '+':
                state = 15;
                break;
            case '-':
                state = 16;
                break;
            case '*':
                state = 17;
                break;
            case '/':
                state = 18;
                break;
            case '%':
                state = 19;
                break;
            case '^':
                state = 20;
                break;
            case '(':
                addSymbol("brace", "(");
                // TODO case '?'
                break;
            case ')':
                addSymbol("brace", ")");
```

```
                break;
            case '{':
                addSymbol("brace", "{");
                break;
            case '}':
                addSymbol("brace", "}");
                break;
            case '[':
                addSymbol("brace", "[");
                break;
            case ']':
                addSymbol("brace", "]");
                break;
            case '\'':
                state = 21;
                break;
            case '"':
                state = 22;
                break;
            case '.':
                addSymbol("postfix operator", ".");
            case ',':
                addSymbol("delimeter", ",");
                break;
            case ';':
                addSymbol("delimeter", ";");
                break;
            case ' ':
            case '  ':
            case '\n':
            case EOF:
                break;
            default:
                stringstream ss;
                ss << "Illegal char \'" << ch << "\' ";
                string err = ss.str();
                logError(err);
                break;
            }
        }
        break;
    case 1: // Identifer state
        if (is_letter() || is_digit()) {
            cat();
            state = 1;
        } else {
            state = 0;
            if (is_keyword()) {
                addSymbol("keyword", bufStr);
            } else {
                addSymbol("identifer", bufStr);
            }
            // Unget last char from buffer.
            unget_char();
            // Clear symbol cache bufStr.
            clrStr();
        }
        break;
```

```
        case 2: // Integer state
            if (is_digit()) {
                cat();
                state = 2;
            } else if (ch == '.') {
                cat();
                state = 3;
            } else if (ch == 'E' || ch == 'e') {
                cat();
                state = 5;
            } else {
                state = 0;
                addSymbol("integer", bufStr);
                unget_char();
                clrStr();
            }
            break;
        case 3: // '.' state
            if (is_digit()) {
                cat();
                state = 4;
            } else {
                state = 0;
                // concat 0 to bufStr
                bufStr.push_back('0');
                addSymbol("float", bufStr);
                unget_char();
                clrStr();
            }
            break;
        case 4: // after '.' state
            if (is_digit()) {
                cat();
                state = 4;
            } else if (ch == 'E' || ch == 'e') {
                // exp state
                cat();
                state = 5;
            } else {
                state = 0;
                addSymbol("float", bufStr);
                unget_char();
                clrStr();
            }
            break;
        case 5:
            if (is_digit()) {
                cat();
                state = 7;
            } else if (ch == '+' || ch == '-') {
                cat();
                state = 6;
            } else {
                state = 0;
                logError("Expected␣exponent");
                unget_char();
                clrStr();
            }
```

```
            break;
        case 6:
            if (is_digit()) {
                cat();
                state = 7;
            } else {
                state = 0;
                logError("Expected␣exponent");
                unget_char();
                clrStr();
            }
            break;
        case 7:
            if (is_digit()) {
                cat();
                state = 7;
            } else {
                state = 0;
                addSymbol("float", bufStr);
                unget_char();
                clrStr();
            }
            break;
        case 8:
            if (ch == '=') {
                addSymbol("relop", "<=");
                state = 0;
            } else if (ch == '<') {
                addSymbol("bitop", "<<");
                state = 0;
            } else {
                addSymbol("relop", "<");
                state = 0;
                unget_char();
            }
            break;
        case 9:
            if (ch == '=') {
                addSymbol("relop", ">=");
                state = 0;
            } else if (ch == '>') {
                addSymbol("bitop", ">>");
                state = 0;
            } else {
                addSymbol("relop", ">");
                state = 0;
                unget_char();
            }
            break;
        case 10:
            if (ch == '=') {
                addSymbol("relop", "==");
                state = 0;
            } else {
                addSymbol("assign-op", "=");
                state = 0;
                unget_char();
            }
```

```
            break;
        case 11:
            if (ch == '=') {
                addSymbol("relop", "!=");
                state = 0;
            } else {
                addSymbol("logic-op", "!");
                state = 0;
                unget_char();
            }
            break;
        case 12:
            if (ch == '=') {
                addSymbol("bitop", "&=");
                state = 0;
            } else if (ch == '&') {
                addSymbol("logic-op", "&&");
                state = 0;
            } else {
                addSymbol("bitop", "&");
                state = 0;
                unget_char();
            }
            break;
        case 13:
            if (ch == '=') {
                addSymbol("bitop", "|=");
                state = 0;
            } else if (ch == '|') {
                addSymbol("logic-op", "||");
                state = 0;
            } else {
                addSymbol("bitop", "|");
                state = 0;
                unget_char();
            }
            break;
        case 14:
            if (ch == '=') {
                addSymbol("bitop", "~=");
                state = 0;
            } else {
                addSymbol("bitop", "~");
                state = 0;
                unget_char();
            }
            break;
        case 15:
            if (ch == '+') {
                addSymbol("arith-op", "++");
                state = 0;
            } else if (ch == '=') {
                addSymbol("arith-op", "+=");
                state = 0;
            } else {
                addSymbol("arith-op", "+");
                state = 0;
                unget_char();
```

```
            }
            break;
        case 16:
            if (ch == '=') {
                addSymbol("arith-op", "-=");
                state = 0;
            } else if (ch == '-') {
                addSymbol("arith-op", "--");
                state = 0;
            } else {
                addSymbol("arith-op", "-");
                state = 0;
                unget_char();
            }
            break;
        case 17:
            if (ch == '=') {
                addSymbol("arith-op", "*=");
                state = 0;
            } else {
                addSymbol("arith-op", "*");
                state = 0;
                unget_char();
            }
            break;
        case 18:
            if (ch == '/') {
                // "//" comment state
                state = 181;
                break;
            } else if (ch == '*') {
                // "/*" comment state
                state = 182;
                break;
            } else if (ch == '=') {
                addSymbol("arith-op", "/=");
                state = 0;
            } else {
                addSymbol("arith-op", "/");
                state = 0;
                unget_char();
            }
            break;
        case 181:
            // state //
            if (ch == '\n') {
                // end of "//" comment line
                state = 0;
            } else {
                state = 181;
            }
            break;
        case 182:
            // state /*
            if (ch == '*') {
                // transfer to state /* *
                state = 183;
            } else {
```

```
            // stay in state /*
            state = 182;
        }
    case 183:
        // state /* *
        if (ch == '/') {
            // state /* */, exit comment state
            state = 0;
        } else if (ch == '*') {
            // state /* ****...., stay in current state
            state = 183;
        } else {
            // fallback to state /*
            state = 182;
        }
        break;
    case 19:
        if (ch == '=') {
            addSymbol("arith-op", "%=");
            state = 0;
        } else {
            addSymbol("arith-op", "%");
            state = 0;
            unget_char();
        }
        break;
    case 20:
        if (ch == '=') {
            addSymbol("bitop", "^=");
            state = 0;
        } else {
            addSymbol("bitop", "^");
            state = 0;
            unget_char();
        }
        break;
    case 21:
        if (ch == '\'') {
            // empty char process
            state = 0;
            logError("Empty char");
            unget_char();
        } else if (ch == '\\') {
            // escape character
            cat();
            state = 21;
        } else {
            cat();
            state = 211; // check whether next char is '
        }
        break;
    case 211:
        if (ch == '\'') {
            // end of char, may or may not be valid
            // valid char, return to state 0
            state = 0;
            addSymbol("char", bufStr);
            clrStr();
```

```
            } else {
                // invalid char, longer than 1 character.
                state = 0;
                logError("Char exceeding the limit length ");
                clrStr();
            }
            break;
        case 22:
            if (ch == '"') {
                // ERROR: empty string
                state = 0;
                logError("Empty string");
            } else {
                state = 221;
                cat();
            }
            break;
        case 221:
            if (ch == '"') {
                // end of string
                state = 0;
                addSymbol("string", bufStr);
                clrStr();
            } else {
                // string is not ended
                state = 221;
                cat();
            }
            break;
        default:
            logError("Unknown state");
            break;
        }
    }
}

void Lex::print() {
    // print number of lines, chars
    cout << endl;
    cout << "======Statistic information======" << endl;
    cout << numLines << " lines, " << numChar << " characters, ";
    // print SymbolList
    sl.print();
    cout << "======End of statistic information======" << endl;
}
} // namespace parser
```

## 5.2 FLEX 版实现

在 C++ 版本实现思路的基础上, 通过学习 FLEX 的语法, 用 FLEX 实现了相同功能的版本.

### 5.2.1 FLEX 介绍

#### 5.2.1.1 基本结构

一个 FLEX 程序的基本结构为

```
Statements/Definitaions
%%
Rules
%%
User Functions(optional)
```

我们将 lexer 保存在扩展名为.i 的文件中.

#### 5.2.1.2 Definitions

我们可以为工具添加如下 **options**:

- %option noyywrap -> flex 将只读一个输入文件
- %option case-insensitive -> flex 将不区分大小写

我们也可以对某些正则表达式设置 **start states**:

```
%x STATE_NAME
```

对于多行注释, 这是很好用的, 因为当到达注释末尾的时候, 可以更方便的搜索连续的内容并结束注释状态.
此外, 我们可以利用正则表达式定义一些 **Identifiers** 并给他们命名, 如:

```
print [ -\~] // space to \~(所有可打印的ASCII字符)
```

语句识别 {print} 作为一个包含所有可打印字符的 group.
使用 identifiers 我们可以在匹配特定 token 的时候方便的使用名称调用, 而不是每次都要书写完整的正则表达式.
最后, 我们也可以定义一个 **literal block of code**, 即事实上的 c 代码, 并且可以包含头文件, 常量, 全局变量和函数定义等. 这些代码将会被复制到 flex 生成的分析器中, 并最终成为编译器的一部分.
要定义一个这样的 block, 要以如下形式:

```
%{
  // literal c code
%}
```

#### 5.2.1.3 Rules

我们在这里定义 tokens 的规则. 我们使用这样的格式:

```
regex-rule { // c action-code }
```

左边部分仅包含正则表达式规则, 而右边部分是定义了行为的实际上的 c 代码. 目前我们将仅仅输出到标准输出说我们找到了一个特定的 token, 在那之后我们会返回 token 的记号和属性.

所以, 要搜索一个可打印字符串序列, 我们可能会使用:

```
{print}+ { printf("Found␣printable␣character␣sequence␣%s\n", yytext); }
```

左边是一条匹配包含至少一个可打印字符的字符串的正则表达式规则, 变量 yytext 包含每次识别到的 token.

#### 5.2.1.4 User Functions

最后, 在 FLEX 中我们也可以定义函数.
例如, 代码可以包含主程序入口 main(), 也可以包含一个错误信息打印函数 yyerror(). yywrap() 函数由 option "%option noyywrap" 定义. 我们可能还需要一个打印 token 的记号和属性的函数, 比如 ret_print(), 将 token 传入到后面的处理程序, 如语法分析程序, 并同时输出到标准输出.

#### 5.2.1.5 Variables

Flex 的变量有如下几种:

- char *yytext -> 包含识别到的 token;
- int yyleng -> 包含识别到的 token 的长度;
- YYSTYPE yylval -> 用来和之后的程序, 例如语法分析程序通信.

#### 5.2.1.6 Usage of Flex

要使用 Flex, 需要先安装 Flex, 之后执行以下命令:

flex lex.l // lex.l is the flex file

clang lex.yy.c -o c_lex -lfl // or use gcc instead, generate c source file

./lex input_file // lex will take input_file as c source file to be analysed

默认情况下, 分析结果将输出到 console 中.

### 5.2.2 FLEX 源代码清单

```
// lex.i
```

```
%option noyywrap

%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    int numLines = 1; // initialize to 1
    void ret_print(char *token_type);
    void yyerror();
%}

%x ML_COMMENT
%x S_CHAR

alpha       [a-zA-Z]
digit       [0-9]
alnum       {alpha}|{digit}
```

```
print       [ -~]

IDENTIFIER          {alpha}+{alnum}*
INTEGER     "0"|[0-9]{digit}*
FLOAT       "0"|{digit}*"."{digit}+
CHAR        ("'"{print}"'")|("\'""\\"[nftrbv0]"\'")
STRING      \".*\"
KEYWORD     "auto"|"switch"|"case"|"for"|"do"|"while"|"int"|"char"|"float"|"double"|"

%%

"//".*                  { ; }

"/*"                    { BEGIN(ML_COMMENT); }
<ML_COMMENT>"*/"        { BEGIN(INITIAL); }
<ML_COMMENT>[^*\n]+
<ML_COMMENT>"*"
<ML_COMMENT>"\n"        { numLines += 1; }


KEYWORD                 { ret_print("keyword");}


"+"|"-"|"*"|"/"|"++"|"--"|"+="|"-="|"*="|"/="|"%"|"%="       { ret_print("arith-op");
"|"|"&"|"~"|"|="|"&="|"~="|"<<"|"<<="|">>"|">>="    { ret_print("bitop"); }
"&&"|"||"                                           { ret_print("logic-op"); }
"="                                                 { ret_print("assign-op"); }
"=="|"!="|"!"|">"|"<"|">="|"<="                     { ret_print("relop"); }


"("|")"|"]"|"["|"{"|"}"     { ret_print("brace"); }
","|";"                     { ret_print("delimeter"); }
"."                         { ret_print("postfix operator"); }


{KEYWORD}       { ret_print("keyword"); }
{IDENTIFIER}    { ret_print("identifier"); }
{INTEGER}       { ret_print("integer"); }
{FLOAT}         { ret_print("float"); }
{CHAR}          { ret_print("char"); }
{STRING}        { ret_print("string"); }


"\n"            { numLines += 1; }
[   \t\r\f]+          /* jump whitespace */

.               { yyerror("Unrecognized character"); }

%%

void ret_print(char *token_type){
    printf("L%d: \t<%s, %s>\n", numLines, token_type, yytext);
}

void yyerror(char *message){
    printf("ERROR: \"%s\" in L%d. Token = %s\n", message, numLines, yytext);
    /* exit(1); */
}
```

```
int main(int argc, char *argv[]){
    yyin = fopen(argv[1], "r");
    yylex();
    fclose(yyin);
    return 0;
}
```

# 6 实验输入 (测试程序)

### 6.0.1 测试样例 1

```
int main() {
  char ch = 'a'; // comment
  char[] str = "hello world"; /* also
  comment*/
  int num = 13;
  printf("%s", str)
}
```

### 6.0.2 测试样例 2

```
typedef struct huf
{int weight;
int parent,lchild,rchild;                    //定义父子
}HTNode,*Huffmantree;

FILE* fptr=NULL;                                                    //初始化文件指针

Huffmantree HT=NULL;                          //哈夫曼树初始化


int filetype(char ch[])                      //文件类型
{int i=0,n;
while(ch[i]!='\0')
        {if(ch[i]=='.')                       //对.的处理
                n=i;
        i++;}
return n;}


void Select(int n,int* s1,int* s2)           //选择
{int i,temp;                                  // s1, s2的处理
for(i=0;i<=n;i++)
        if(HT[i].weight>0&&HT[i].parent==-1)  //权重比较和处理
                {*s1=i;
                 break;}
for(i=i+1;i<=n;i++)
        {if(HT[i].weight>0&&HT[i].parent==-1)
                {*s2=i;
                 break;}
        }
if(HT[*s2].weight<HT[*s1].weight)
```

```
        {temp=*s1;
         *s1=*s2;
         *s2=temp;
        }
for(i=i+1;i<=n;i++)
        if(HT[i].weight>0&&HT[i].parent==-1)
                if(HT[i].weight<HT[*s1].weight)
                        {*s2=*s1;
                        *s1=i;}
                else if(HT[i].weight<HT[*s2].weight)
                        *s2=i;
return;
}


int CreateHuffmantree(char filename[])                                  //创建哈夫曼树
{int i,k,n=0;
 int s1,s2;
HT=(Huffmantree)malloc(sizeof(HTNode)*512);
if((fptr=fopen(filename,"rb"))==NULL)
        {printf("can't␣open␣the␣source␣file\n");
        return 0;}
for(i=0;i<=255;i++)
        {HT[i].parent=HT[i].lchild=HT[i].rchild=-1;
        HT[i].weight=0;}
while((k=fgetc(fptr))!=EOF)
        {if(HT[k].weight==0)
                n++;
         HT[k].weight++;
        }
rewind(fptr);
for(i=256;i<256+n-1;i++)
        {Select(i-1,&s1,&s2);
        HT[s1].parent=i;
        HT[s2].parent=i;
        HT[i].lchild=s1;
        HT[i].rchild=s2;
        HT[i].weight=HT[s1].weight+HT[s2].weight;
        HT[i].parent=-1;
        }
return n;
}


char** CreateHuffmancode(int n)
{int i,j,start,pre;
char**HC=(char**)malloc(sizeof(char*)*256);
for(i=0;i<256;i++)
        HC[i]=(char*)malloc(sizeof(char)*(n+1));
char cd[n];
cd[n-1]='\0';

for(i=0;i<=255;i++)
        {       if(HT[i].weight==0)
                        HC[i]='\0';
                else
                        {start=n-1;
                        j=i;
```

```
                        pre=HT[j].parent;
                        while(pre!=-1)
                                {start--;
                                if(HT[pre].lchild==j)
                                        cd[start]='0';
                                else
                                        cd[start]='1';
                                j=pre;
                                pre=HT[j].parent;
                                }
                        strcpy(HC[i],&cd[start]);
                        }
        }
return HC;
}

void  Huffmandecoding(FILE* fptr1,int num,int pos,char*filename,int k)
{rewind(fptr1);
 FILE* fptr2;
 char objectfile[10]="out2";
strcat(objectfile,&filename[k]);;
if((fptr2=fopen(objectfile,"wb"))==NULL)
        printf("can't␣open␣the␣object␣file\n");
int c;
int i,j,m,p,weight;
for(i=0;i<=512;i++)
        if((HT[i].parent==-1)&&HT[i].weight>0)
                        break;
p=i;
while((c=fgetc(fptr1))!=EOF)
                {num--;
                for(weight=128;weight>0;weight=weight/2)
                        {m=c/weight;
                        c=c%weight;
                        if(m==0)
                                p=HT[p].lchild;
                        if(m==1)
                                p=HT[p].rchild;
                        if(HT[p].lchild==-1&&HT[p].rchild==-1)
                                {fputc(p,fptr2);
                                p=i;
                                if(num==-1&&weight==pos*2)
                                        break;}
                        }
                }
printf("哈夫曼编码结束,␣输出到文件");
fclose(fptr2);
return;
}

int main(void)
{int n,i,j,k,weight,pos,num=0;
char c,ch;
char **HC;
FILE* fptr0;
FILE* fptr1;
if((fptr0=fopen("out0.txt","w+"))==NULL)
        printf("can't␣open␣the␣object␣file\n");
```

```
if((fptr1=fopen("out1.txt","wb+")) ==NULL)
        printf("can't␣open␣the␣object␣file\n");
printf("对象文件无法打开");
char filename[100];
gets(filename);
n=CreateHuffmantree(filename);
if(n==0)
        return 0;
if(n==1)
        {rewind(fptr);
        while((c=fgetc(fptr))!=EOF)
                {k++;
                ch=c;
                fputc('0',fptr0);
                if(k%8==0)
                        fputc((char)0,fptr1); }
        fputc((char)0,fptr1);}

else
{HC=CreateHuffmancode(n);                //创建哈夫曼代码

while((i=fgetc(fptr))!=EOF)
        fputs(HC[i],fptr0);                        //写入字符串到文件


rewind(fptr0);
weight=128;
j=0;
while((i=fgetc(fptr0))!=EOF)
        {j+=(i-48)*weight;
        weight=weight/2;
         if(weight==0)
                {weight=128;
                fputc((unsigned char)j,fptr1);
                num++;
                j=0;}
        }
if(weight!=0)
        {fputc(j,fptr1);
        pos=weight;      }    }
printf("处理过程如下:\n");
c=getchar();
if(c=='y')
        if(n>1)
        {
                Huffmandecoding(fptr1,num,pos,filename,filetype(filename));
                free(HT);
        for(i=0;i<256;i++)
                free(HC[i]);
        free(HC);}
        else
        {       rewind(fptr1);
                FILE* fptr2;
                char objectfile[10]="out2";
                strcat(objectfile,&filename[filetype(filename)]);;
                if((fptr2=fopen(objectfile,"wb")) ==NULL)
                        printf("can't␣open␣the␣object␣file\n");
                for(i=0;i<k;i++)
```

```
                    fputc(ch,fptr2);
                printf("处理结果写入到out2中");
                fclose(fptr2);}


fclose(fptr);
fclose(fptr0);
fclose(fptr1);
return 0;
}
```

# 7 实验运行结果及分析说明

## 7.1 输出格式概述

- 在 Statistic information 之前的是输入词法分析程序的源代码文件中解析到的符号, 格式为 L[行号]: <[记号], [属性]>.
- Statistic information(统计信息) 有如下格式:
  - 第一行为行数, 字符数和符号数统计, 格式为 [行数] lines, [字符数] characters, [符号数] symbols;
  - 之后跟 [符号数] 行, 每行是一个符号及其出现次数, 格式为 <[记号], [属性]> appeared [出现次数] times;

## 7.2 测试样例 1 的输出

测试样例 1 为较简短的样例, 用于进行输入输出功能和状态转移基本功能的测试;

### 7.2.1 C++ 版词法分析程序测试

```
L1: <keyword, int>
L1: <identifer, main>
L1: <block_symbol, (>
L1: <block_symbol, )>
L1: <block_symbol, {>
L2: <keyword, char>
L2: <identifer, ch>
L2: <assign-op, =>
L2: <char, a>
L2: <delimeter, ;>
L3: <keyword, char>
L3: <block_symbol, [>
L3: <block_symbol, ]>
L3: <identifer, str>
L3: <assign-op, =>
L3: <string, hello world>
L3: <delimeter, ;>
L5: <keyword, int>
L5: <identifer, num>
```

```
L5: <assign-op, =>
L5: <integer, 13>
L5: <delimeter, ;>
L6: <identifer, printf>
L6: <block_symbol, (>
L6: <string, %s>
L6: <delimeter, ,>
L6: <identifer, str>
L6: <block_symbol, )>
L7: <block_symbol, }>

======Statistic information======
7 lines, 129 characters, 20 symbols
<keyword, int> appeared 2 times
<identifer, main> appeared 1 times
<block_symbol, (> appeared 2 times
<block_symbol, )> appeared 2 times
<block_symbol, {> appeared 1 times
<keyword, char> appeared 2 times
<identifer, ch> appeared 1 times
<assign-op, => appeared 3 times
<char, a> appeared 1 times
<delimeter, ;> appeared 3 times
<block_symbol, [> appeared 1 times
<block_symbol, ]> appeared 1 times
<identifer, str> appeared 2 times
<string, hello world> appeared 1 times
<identifer, num> appeared 1 times
<integer, 13> appeared 1 times
<identifer, printf> appeared 1 times
<string, %s> appeared 1 times
<delimeter, ,> appeared 1 times
<block_symbol, }> appeared 1 times
======End of statistic information======
```

### 7.2.2 FLEX 版词法分析程序测试

```
L1:     <keyword, int>
L1:     <identifier, main>
L1:     <brace, (>
L1:     <brace, )>
L1:     <brace, {>
L2:     <keyword, char>
L2:     <identifier, ch>
L2:     <assign-op, =>
L2:     <char, 'a'>
L2:     <delimeter, ;>
Jumped single line comment at line 2
L3:     <keyword, char>
L3:     <brace, [>
L3:     <brace, ]>
L3:     <identifier, str>
L3:     <assign-op, =>
L3:     <string, "hello␣world">
L3:     <delimeter, ;>
Jumped multi-line comment from line 3 to line 4
```

```
L5:     <keyword, int>
L5:     <identifier, num>
L5:     <assign-op, =>
L5:     <integer, 13>
L5:     <delimeter, ;>
L6:     <identifier, printf>
L6:     <brace, (>
L6:     <string, "%s">
L6:     <delimeter, ,>
L6:     <identifier, str>
L6:     <brace, )>
L7:     <brace, }>
```

## 7.3 测试样例 2 的输出

测试样例 2 为较长, 测试内容较全面的样例, 用于对词法分析程序实现的各项功能进行测试.

### 7.3.1 C++ 版词法分析程序测试

```
L1: <keyword, typedef>
L1: <keyword, struct>
L2: <identifer, huf>
L2: <brace, {>
L2: <keyword, int>
L2: <identifer, weight>
L2: <delimeter, ;>
L3: <keyword, int>
L3: <identifer, parent>
L3: <delimeter, ,>
L3: <identifer, lchild>
L3: <delimeter, ,>
L3: <identifer, rchild>
L3: <delimeter, ;>
L4: <brace, }>
L4: <identifer, HTNode>
L4: <delimeter, ,>
L4: <arith-op, *>
L4: <identifer, Huffmantree>
L4: <delimeter, ;>
L6: <identifer, FILE>
L6: <arith-op, *>
L6: <identifer, fptr>
L6: <assign-op, =>
L6: <identifer, NULL>
L6: <delimeter, ;>
L8: <identifer, Huffmantree>
L8: <identifer, HT>
L8: <assign-op, =>
L8: <identifer, NULL>
L8: <delimeter, ;>
L11: <keyword, int>
L11: <identifer, filetype>
L11: <brace, (>
```

```
L11: <keyword, char>
L11: <identifer, ch>
L11: <brace, [>
L11: <brace, ]>
L11: <brace, )>
L12: <brace, {>
L12: <keyword, int>
L12: <identifer, i>
L12: <assign-op, =>
L12: <delimeter, ,>
L12: <identifer, n>
L12: <delimeter, ;>
L13: <keyword, while>
L13: <brace, (>
L13: <identifer, ch>
L13: <brace, [>
L13: <identifer, i>
L13: <brace, ]>
L13: <relop, !=>
L13: <char, \0>
L13: <brace, )>
L14: <brace, {>
L14: <keyword, if>
L14: <brace, (>
L14: <identifer, ch>
L14: <brace, [>
L14: <identifer, i>
L14: <brace, ]>
L14: <relop, ==>
L14: <char, .>
L14: <brace, )>
L15: <identifer, n>
L15: <assign-op, =>
L15: <identifer, i>
L15: <delimeter, ;>
L16: <identifer, i>
L16: <arith-op, ++>
L16: <delimeter, ;>
L16: <brace, }>
L17: <keyword, return>
L17: <identifer, n>
L17: <delimeter, ;>
L17: <brace, }>
L20: <keyword, void>
L20: <identifer, Select>
L20: <brace, (>
L20: <keyword, int>
L20: <identifer, n>
L20: <delimeter, ,>
L20: <keyword, int>
L20: <arith-op, *>
L20: <identifer, s1>
L20: <delimeter, ,>
L20: <keyword, int>
L20: <arith-op, *>
L20: <identifer, s2>
L20: <brace, )>
L21: <brace, {>
```

```
L21: <keyword, int>
L21: <identifer, i>
L21: <delimeter, ,>
L21: <identifer, temp>
L21: <delimeter, ;>
L22: <keyword, for>
L22: <brace, (>
L22: <identifer, i>
L22: <assign-op, =>
L22: <delimeter, ;>
L22: <identifer, i>
L22: <relop, <=>
L22: <identifer, n>
L22: <delimeter, ;>
L22: <identifer, i>
L22: <arith-op, ++>
L22: <brace, )>
L23: <keyword, if>
L23: <brace, (>
L23: <identifer, HT>
L23: <brace, [>
L23: <identifer, i>
L23: <brace, ]>
L23: <postfix operator, .>
L23: <delimeter, ,>
L23: <identifer, weight>
L23: <relop, >>
L23: <logic-op, &&>
L23: <identifer, HT>
L23: <brace, [>
L23: <identifer, i>
L23: <brace, ]>
L23: <postfix operator, .>
L23: <delimeter, ,>
L23: <identifer, parent>
L23: <relop, ==>
L23: <arith-op, ->
L23: <integer, 1>
L23: <brace, )>
L24: <brace, {>
L24: <arith-op, *>
L24: <identifer, s1>
L24: <assign-op, =>
L24: <identifer, i>
L24: <delimeter, ;>
L25: <keyword, break>
L25: <delimeter, ;>
L25: <brace, }>
L26: <keyword, for>
L26: <brace, (>
L26: <identifer, i>
L26: <assign-op, =>
L26: <identifer, i>
L26: <arith-op, +>
L26: <integer, 1>
L26: <delimeter, ;>
L26: <identifer, i>
L26: <relop, <=>
```

```
L26: <identifer, n>
L26: <delimeter, ;>
L26: <identifer, i>
L26: <arith-op, ++>
L26: <brace, )>
L27: <brace, {>
L27: <keyword, if>
L27: <brace, (>
L27: <identifer, HT>
L27: <brace, [>
L27: <identifer, i>
L27: <brace, ]>
L27: <postfix operator, .>
L27: <delimeter, ,>
L27: <identifer, weight>
L27: <relop, >>
L27: <logic-op, &&>
L27: <identifer, HT>
L27: <brace, [>
L27: <identifer, i>
L27: <brace, ]>
L27: <postfix operator, .>
L27: <delimeter, ,>
L27: <identifer, parent>
L27: <relop, ==>
L27: <arith-op, ->
L27: <integer, 1>
L27: <brace, )>
L28: <brace, {>
L28: <arith-op, *>
L28: <identifer, s2>
L28: <assign-op, =>
L28: <identifer, i>
L28: <delimeter, ;>
L29: <keyword, break>
L29: <delimeter, ;>
L29: <brace, }>
L30: <brace, }>
L31: <keyword, if>
L31: <brace, (>
L31: <identifer, HT>
L31: <brace, [>
L31: <arith-op, *>
L31: <identifer, s2>
L31: <brace, ]>
L31: <postfix operator, .>
L31: <delimeter, ,>
L31: <identifer, weight>
L31: <relop, <>
L31: <identifer, HT>
L31: <brace, [>
L31: <arith-op, *>
L31: <identifer, s1>
L31: <brace, ]>
L31: <postfix operator, .>
L31: <delimeter, ,>
L31: <identifer, weight>
L31: <brace, )>
```

```
L32: <brace , {>
L32: <identifer , temp>
L32: <assign-op, =>
L32: <arith-op, *>
L32: <identifer , s1>
L32: <delimeter , ;>
L33: <arith-op, *>
L33: <identifer , s1>
L33: <assign-op, =>
L33: <arith-op, *>
L33: <identifer , s2>
L33: <delimeter , ;>
L34: <arith-op, *>
L34: <identifer , s2>
L34: <assign-op, =>
L34: <identifer , temp>
L34: <delimeter , ;>
L35: <brace , }>
L36: <keyword, for>
L36: <brace , (>
L36: <identifer , i>
L36: <assign-op, =>
L36: <identifer , i>
L36: <arith-op, +>
L36: <integer , 1>
L36: <delimeter , ;>
L36: <identifer , i>
L36: <relop , <=>
L36: <identifer , n>
L36: <delimeter , ;>
L36: <identifer , i>
L36: <arith-op, ++>
L36: <brace , )>
L37: <keyword, if>
L37: <brace , (>
L37: <identifer , HT>
L37: <brace , [>
L37: <identifer , i>
L37: <brace , ]>
L37: <postfix operator, .>
L37: <delimeter , ,>
L37: <identifer , weight>
L37: <relop , >>
L37: <logic-op, &&>
L37: <identifer , HT>
L37: <brace , [>
L37: <identifer , i>
L37: <brace , ]>
L37: <postfix operator, .>
L37: <delimeter , ,>
L37: <identifer , parent>
L37: <relop , ==>
L37: <arith-op, ->
L37: <integer , 1>
L37: <brace , )>
L38: <keyword, if>
L38: <brace , (>
L38: <identifer , HT>
```

```
L38: <brace, [>
L38: <identifer, i>
L38: <brace, ]>
L38: <postfix operator, .>
L38: <delimeter, ,>
L38: <identifer, weight>
L38: <relop, <>
L38: <identifer, HT>
L38: <brace, [>
L38: <arith-op, *>
L38: <identifer, s1>
L38: <brace, ]>
L38: <postfix operator, .>
L38: <delimeter, ,>
L38: <identifer, weight>
L38: <brace, )>
L39: <brace, {>
L39: <arith-op, *>
L39: <identifer, s2>
L39: <assign-op, =>
L39: <arith-op, *>
L39: <identifer, s1>
L39: <delimeter, ;>
L40: <arith-op, *>
L40: <identifer, s1>
L40: <assign-op, =>
L40: <identifer, i>
L40: <delimeter, ;>
L40: <brace, }>
L41: <keyword, else>
L41: <keyword, if>
L41: <brace, (>
L41: <identifer, HT>
L41: <brace, [>
L41: <identifer, i>
L41: <brace, ]>
L41: <postfix operator, .>
L41: <delimeter, ,>
L41: <identifer, weight>
L41: <relop, <>
L41: <identifer, HT>
L41: <brace, [>
L41: <arith-op, *>
L41: <identifer, s2>
L41: <brace, ]>
L41: <postfix operator, .>
L41: <delimeter, ,>
L41: <identifer, weight>
L41: <brace, )>
L42: <arith-op, *>
L42: <identifer, s2>
L42: <assign-op, =>
L42: <identifer, i>
L42: <delimeter, ;>
L43: <keyword, return>
L43: <delimeter, ;>
L44: <brace, }>
L47: <keyword, int>
```

```
L47: <identifer, CreateHuffmantree>
L47: <brace, (>
L47: <keyword, char>
L47: <identifer, filename>
L47: <brace, [>
L47: <brace, ]>
L47: <brace, )>
L48: <brace, {>
L48: <keyword, int>
L48: <identifer, i>
L48: <delimeter, ,>
L48: <identifer, k>
L48: <delimeter, ,>
L48: <identifer, n>
L48: <assign-op, =>
L48: <delimeter, ;>
L49: <keyword, int>
L49: <identifer, s1>
L49: <delimeter, ,>
L49: <identifer, s2>
L49: <delimeter, ;>
L50: <identifer, HT>
L50: <assign-op, =>
L50: <brace, (>
L50: <identifer, Huffmantree>
L50: <brace, )>
L50: <identifer, malloc>
L50: <brace, (>
L50: <keyword, sizeof>
L50: <brace, (>
L50: <identifer, HTNode>
L50: <brace, )>
L50: <arith-op, *>
L50: <integer, 512>
L50: <brace, )>
L50: <delimeter, ;>
L51: <keyword, if>
L51: <brace, (>
L51: <brace, (>
L51: <identifer, fptr>
L51: <assign-op, =>
L51: <identifer, fopen>
L51: <brace, (>
L51: <identifer, filename>
L51: <delimeter, ,>
L51: <string, rb>
L51: <brace, )>
L51: <brace, )>
L51: <relop, ==>
L51: <identifer, NULL>
L51: <brace, )>
L52: <brace, {>
L52: <identifer, printf>
L52: <brace, (>
L52: <string, can't␣open␣the␣source␣file\n>
L52:␣<brace,␣)>
L52:␣<delimeter,␣;>
L53:␣<keyword,␣return>
```

```
L53: <delimeter , ;>
L53: <brace , }>
L54: <keyword , for>
L54: <brace , (>
L54: <identifer , i>
L54: <assign-op , =>
L54: <delimeter , ;>
L54: <identifer , i>
L54: <relop , <=>
L54: <integer , 255>
L54: <delimeter , ;>
L54: <identifer , i>
L54: <arith-op , ++>
L54: <brace , )>
L55: <brace , {>
L55: <identifer , HT>
L55: <brace , [>
L55: <identifer , i>
L55: <brace , ]>
L55: <postfix operator , .>
L55: <delimeter , ,>
L55: <identifer , parent>
L55: <assign-op , =>
L55: <identifer , HT>
L55: <brace , [>
L55: <identifer , i>
L55: <brace , ]>
L55: <postfix operator , .>
L55: <delimeter , ,>
L55: <identifer , lchild>
L55: <assign-op , =>
L55: <identifer , HT>
L55: <brace , [>
L55: <identifer , i>
L55: <brace , ]>
L55: <postfix operator , .>
L55: <delimeter , ,>
L55: <identifer , rchild>
L55: <assign-op , =>
L55: <arith-op , ->
L55: <integer , 1>
L55: <delimeter , ;>
L56: <identifer , HT>
L56: <brace , [>
L56: <identifer , i>
L56: <brace , ]>
L56: <postfix operator , .>
L56: <delimeter , ,>
L56: <identifer , weight>
L56: <assign-op , =>
L56: <delimeter , ;>
L56: <brace , }>
L57: <keyword , while>
L57: <brace , (>
L57: <brace , (>
L57: <identifer , k>
L57: <assign-op , =>
L57: <identifer , fgetc>
```

```
L57: <brace , (>
L57: <identifer , fptr>
L57: <brace , )>
L57: <brace , )>
L57: <relop , !=>
L57: <identifer , EOF>
L57: <brace , )>
L58: <brace , {>
L58: <keyword , if>
L58: <brace , (>
L58: <identifer , HT>
L58: <brace , [>
L58: <identifer , k>
L58: <brace , ]>
L58: <postfix operator , .>
L58: <delimeter , ,>
L58: <identifer , weight>
L58: <relop , ==>
L58: <brace , )>
L59: <identifer , n>
L59: <arith -op , ++>
L59: <delimeter , ;>
L60: <identifer , HT>
L60: <brace , [>
L60: <identifer , k>
L60: <brace , ]>
L60: <postfix operator , .>
L60: <delimeter , ,>
L60: <identifer , weight>
L60: <arith -op , ++>
L60: <delimeter , ;>
L61: <brace , }>
L62: <identifer , rewind>
L62: <brace , (>
L62: <identifer , fptr>
L62: <brace , )>
L62: <delimeter , ;>
L63: <keyword , for>
L63: <brace , (>
L63: <identifer , i>
L63: <assign -op , =>
L63: <integer , 256>
L63: <delimeter , ;>
L63: <identifer , i>
L63: <relop , <>
L63: <integer , 256>
L63: <arith -op , +>
L63: <identifer , n>
L63: <arith -op , ->
L63: <integer , 1>
L63: <delimeter , ;>
L63: <identifer , i>
L63: <arith -op , ++>
L63: <brace , )>
L64: <brace , {>
L64: <identifer , Select>
L64: <brace , (>
L64: <identifer , i>
```

```
L64:␣<arith-op,␣->
L64:␣<integer,␣1>
L64:␣<delimeter,␣,>
L64:␣<bitop,␣&>
L64:␣<identifer,␣s1>
L64:␣<delimeter,␣,>
L64:␣<bitop,␣&>
L64:␣<identifer,␣s2>
L64:␣<brace,␣)>
L64:␣<delimeter,␣;>
L65:␣<identifer,␣HT>
L65:␣<brace,␣[>
L65:␣<identifer,␣s1>
L65:␣<brace,␣]>
L65:␣<postfix␣operator,␣.>
L65:␣<delimeter,␣,>
L65:␣<identifer,␣parent>
L65:␣<assign-op,␣=>
L65:␣<identifer,␣i>
L65:␣<delimeter,␣;>
L66:␣<identifer,␣HT>
L66:␣<brace,␣[>
L66:␣<identifer,␣s2>
L66:␣<brace,␣]>
L66:␣<postfix␣operator,␣.>
L66:␣<delimeter,␣,>
L66:␣<identifer,␣parent>
L66:␣<assign-op,␣=>
L66:␣<identifer,␣i>
L66:␣<delimeter,␣;>
L67:␣<identifer,␣HT>
L67:␣<brace,␣[>
L67:␣<identifer,␣i>
L67:␣<brace,␣]>
L67:␣<postfix␣operator,␣.>
L67:␣<delimeter,␣,>
L67:␣<identifer,␣lchild>
L67:␣<assign-op,␣=>
L67:␣<identifer,␣s1>
L67:␣<delimeter,␣;>
L68:␣<identifer,␣HT>
L68:␣<brace,␣[>
L68:␣<identifer,␣i>
L68:␣<brace,␣]>
L68:␣<postfix␣operator,␣.>
L68:␣<delimeter,␣,>
L68:␣<identifer,␣rchild>
L68:␣<assign-op,␣=>
L68:␣<identifer,␣s2>
L68:␣<delimeter,␣;>
L69:␣<identifer,␣HT>
L69:␣<brace,␣[>
L69:␣<identifer,␣i>
L69:␣<brace,␣]>
L69:␣<postfix␣operator,␣.>
L69:␣<delimeter,␣,>
L69:␣<identifer,␣weight>
L69:␣<assign-op,␣=>
```

```
L69: <identifer , HT>
L69: <brace , [>
L69: <identifer , s1>
L69: <brace , ]>
L69: <postfix operator , .>
L69: <delimeter , ,>
L69: <identifer , weight>
L69: <arith-op , +>
L69: <identifer , HT>
L69: <brace , [>
L69: <identifer , s2>
L69: <brace , ]>
L69: <postfix operator , .>
L69: <delimeter , ,>
L69: <identifer , weight>
L69: <delimeter , ;>
L70: <identifer , HT>
L70: <brace , [>
L70: <identifer , i>
L70: <brace , ]>
L70: <postfix operator , .>
L70: <delimeter , ,>
L70: <identifer , parent>
L70: <assign-op , =>
L70: <arith-op , ->
L70: <integer , 1>
L70: <delimeter , ;>
L71: <brace , }>
L72: <keyword , return>
L72: <identifer , n>
L72: <delimeter , ;>
L73: <brace , }>
L76: <keyword , char>
L76: <arith-op , *>
L76: <arith-op , *>
L76: <identifer , CreateHuffmancode>
L76: <brace , (>
L76: <keyword , int>
L76: <identifer , n>
L76: <brace , )>
L77: <brace , {>
L77: <keyword , int>
L77: <identifer , i>
L77: <delimeter , ,>
L77: <identifer , j>
L77: <delimeter , ,>
L77: <identifer , start>
L77: <delimeter , ,>
L77: <identifer , pre>
L77: <delimeter , ;>
L78: <keyword , char>
L78: <arith-op , *>
L78: <arith-op , *>
L78: <identifer , HC>
L78: <assign-op , =>
L78: <brace , (>
L78: <keyword , char>
L78: <arith-op , *>
```

```
L78: <arith-op, *>
L78: <brace, )>
L78: <identifer, malloc>
L78: <brace, (>
L78: <keyword, sizeof>
L78: <brace, (>
L78: <keyword, char>
L78: <arith-op, *>
L78: <brace, )>
L78: <arith-op, *>
L78: <integer, 256>
L78: <brace, )>
L78: <delimeter, ;>
L79: <keyword, for>
L79: <brace, (>
L79: <identifer, i>
L79: <assign-op, =>
L79: <delimeter, ;>
L79: <identifer, i>
L79: <relop, <>
L79: <integer, 256>
L79: <delimeter, ;>
L79: <identifer, i>
L79: <arith-op, ++>
L79: <brace, )>
L80: <identifer, HC>
L80: <brace, [>
L80: <identifer, i>
L80: <brace, ]>
L80: <assign-op, =>
L80: <brace, (>
L80: <keyword, char>
L80: <arith-op, *>
L80: <brace, )>
L80: <identifer, malloc>
L80: <brace, (>
L80: <keyword, sizeof>
L80: <brace, (>
L80: <keyword, char>
L80: <brace, )>
L80: <arith-op, *>
L80: <brace, (>
L80: <identifer, n>
L80: <arith-op, +>
L80: <integer, 1>
L80: <brace, )>
L80: <brace, )>
L80: <delimeter, ;>
L81: <keyword, char>
L81: <identifer, cd>
L81: <brace, [>
L81: <identifer, n>
L81: <brace, ]>
L81: <delimeter, ;>
L82: <identifer, cd>
L82: <brace, [>
L82: <identifer, n>
L82: <arith-op, ->
```

```
L82: <integer, 1>
L82: <brace, ]>
L82: <assign-op, =>
L82: <char, \0>
L82: <delimeter, ;>
L84: <keyword, for>
L84: <brace, (>
L84: <identifer, i>
L84: <assign-op, =>
L84: <delimeter, ;>
L84: <identifer, i>
L84: <relop, <=>
L84: <integer, 255>
L84: <delimeter, ;>
L84: <identifer, i>
L84: <arith-op, ++>
L84: <brace, )>
L85: <brace, {>
L85: <keyword, if>
L85: <brace, (>
L85: <identifer, HT>
L85: <brace, [>
L85: <identifer, i>
L85: <brace, ]>
L85: <postfix operator, .>
L85: <delimeter, ,>
L85: <identifer, weight>
L85: <relop, ==>
L85: <brace, )>
L86: <identifer, HC>
L86: <brace, [>
L86: <identifer, i>
L86: <brace, ]>
L86: <assign-op, =>
L86: <char, \0>
L86: <delimeter, ;>
L88: <keyword, else>
L88: <brace, {>
L88: <identifer, start>
L88: <assign-op, =>
L88: <identifer, n>
L88: <arith-op, ->
L88: <integer, 1>
L88: <delimeter, ;>
L89: <identifer, j>
L89: <assign-op, =>
L89: <identifer, i>
L89: <delimeter, ;>
L90: <identifer, pre>
L90: <assign-op, =>
L90: <identifer, HT>
L90: <brace, [>
L90: <identifer, j>
L90: <brace, ]>
L90: <postfix operator, .>
L90: <delimeter, ,>
L90: <identifer, parent>
L90: <delimeter, ;>
```

```
L91: <keyword, while>
L91: <brace, (>
L91: <identifer, pre>
L91: <relop, !=>
L91: <arith-op, ->
L91: <integer, 1>
L91: <brace, )>
L92: <brace, {>
L92: <identifer, start>
L92: <arith-op, -->
L92: <delimeter, ;>
L93: <keyword, if>
L93: <brace, (>
L93: <identifer, HT>
L93: <brace, [>
L93: <identifer, pre>
L93: <brace, ]>
L93: <postfix operator, .>
L93: <delimeter, ,>
L93: <identifer, lchild>
L93: <relop, ==>
L93: <identifer, j>
L93: <brace, )>
L94: <identifer, cd>
L94: <brace, [>
L94: <identifer, start>
L94: <brace, ]>
L94: <assign-op, =>
L94: <char, 0>
L94: <delimeter, ;>
L96: <keyword, else>
L96: <identifer, cd>
L96: <brace, [>
L96: <identifer, start>
L96: <brace, ]>
L96: <assign-op, =>
L96: <char, 1>
L96: <delimeter, ;>
L97: <identifer, j>
L97: <assign-op, =>
L97: <identifer, pre>
L97: <delimeter, ;>
L98: <identifer, pre>
L98: <assign-op, =>
L98: <identifer, HT>
L98: <brace, [>
L98: <identifer, j>
L98: <brace, ]>
L98: <postfix operator, .>
L98: <delimeter, ,>
L98: <identifer, parent>
L98: <delimeter, ;>
L99: <brace, }>
L100: <identifer, strcpy>
L100: <brace, (>
L100: <identifer, HC>
L100: <brace, [>
L100: <identifer, i>
```

```
L100: <brace, ]>
L100: <delimeter, ,>
L100: <bitop, &>
L100: <identifer, cd>
L100: <brace, [>
L100: <identifer, start>
L100: <brace, ]>
L100: <brace, )>
L100: <delimeter, ;>
L101: <brace, }>
L102: <brace, }>
L103: <keyword, return>
L103: <identifer, HC>
L103: <delimeter, ;>
L104: <brace, }>
L106: <keyword, void>
L106: <identifer, Huffmandecoding>
L106: <brace, (>
L106: <identifer, FILE>
L106: <arith-op, *>
L106: <identifer, fptr1>
L106: <delimeter, ,>
L106: <keyword, int>
L106: <identifer, num>
L106: <delimeter, ,>
L106: <keyword, int>
L106: <identifer, pos>
L106: <delimeter, ,>
L106: <keyword, char>
L106: <arith-op, *>
L106: <identifer, filename>
L106: <delimeter, ,>
L106: <keyword, int>
L106: <identifer, k>
L106: <brace, )>
L107: <brace, {>
L107: <identifer, rewind>
L107: <brace, (>
L107: <identifer, fptr1>
L107: <brace, )>
L107: <delimeter, ;>
L108: <identifer, FILE>
L108: <arith-op, *>
L108: <identifer, fptr2>
L108: <delimeter, ;>
L109: <keyword, char>
L109: <identifer, objectfile>
L109: <brace, [>
L109: <integer, 10>
L109: <brace, ]>
L109: <assign-op, =>
L109: <string, out2>
L109: <delimeter, ;>
L110: <identifer, strcat>
L110: <brace, (>
L110: <identifer, objectfile>
L110: <delimeter, ,>
L110: <bitop, &>
```

```
L110:␣<identifer,␣filename>
L110:␣<brace,␣[>
L110:␣<identifer,␣k>
L110:␣<brace,␣]>
L110:␣<brace,␣)>
L110:␣<delimeter,␣;>
L110:␣<delimeter,␣;>
L111:␣<keyword,␣if>
L111:␣<brace,␣(>
L111:␣<brace,␣(>
L111:␣<identifer,␣fptr2>
L111:␣<assign-op,␣=>
L111:␣<identifer,␣fopen>
L111:␣<brace,␣(>
L111:␣<identifer,␣objectfile>
L111:␣<delimeter,␣,>
L111:␣<string,␣wb>
L111:␣<brace,␣)>
L111:␣<brace,␣)>
L111:␣<relop,␣==>
L111:␣<identifer,␣NULL>
L111:␣<brace,␣)>
L112:␣<identifer,␣printf>
L112:␣<brace,␣(>
L112:␣<string,␣can't open the object file\n>
L112: <brace, )>
L112: <delimeter, ;>
L113: <keyword, int>
L113: <identifer, c>
L113: <delimeter, ;>
L114: <keyword, int>
L114: <identifer, i>
L114: <delimeter, ,>
L114: <identifer, j>
L114: <delimeter, ,>
L114: <identifer, m>
L114: <delimeter, ,>
L114: <identifer, p>
L114: <delimeter, ,>
L114: <identifer, weight>
L114: <delimeter, ;>
L115: <keyword, for>
L115: <brace, (>
L115: <identifer, i>
L115: <assign-op, =>
L115: <delimeter, ;>
L115: <identifer, i>
L115: <relop, <=>
L115: <integer, 512>
L115: <delimeter, ;>
L115: <identifer, i>
L115: <arith-op, ++>
L115: <brace, )>
L116: <keyword, if>
L116: <brace, (>
L116: <brace, (>
L116: <identifer, HT>
L116: <brace, [>
```

```
L116: <identifer, i>
L116: <brace, ]>
L116: <postfix operator, .>
L116: <delimeter, ,>
L116: <identifer, parent>
L116: <relop, ==>
L116: <arith-op, ->
L116: <integer, 1>
L116: <brace, )>
L116: <logic-op, &&>
L116: <identifer, HT>
L116: <brace, [>
L116: <identifer, i>
L116: <brace, ]>
L116: <postfix operator, .>
L116: <delimeter, ,>
L116: <identifer, weight>
L116: <relop, >>
L116: <brace, )>
L117: <keyword, break>
L117: <delimeter, ;>
L118: <identifer, p>
L118: <assign-op, =>
L118: <identifer, i>
L118: <delimeter, ;>
L119: <keyword, while>
L119: <brace, (>
L119: <brace, (>
L119: <identifer, c>
L119: <assign-op, =>
L119: <identifer, fgetc>
L119: <brace, (>
L119: <identifer, fptr1>
L119: <brace, )>
L119: <brace, )>
L119: <relop, !=>
L119: <identifer, EOF>
L119: <brace, )>
L120: <brace, {>
L120: <identifer, num>
L120: <arith-op, -->
L120: <delimeter, ;>
L121: <keyword, for>
L121: <brace, (>
L121: <identifer, weight>
L121: <assign-op, =>
L121: <integer, 128>
L121: <delimeter, ;>
L121: <identifer, weight>
L121: <relop, >>
L121: <delimeter, ;>
L121: <identifer, weight>
L121: <assign-op, =>
L121: <identifer, weight>
L121: <arith-op, />
L121: <integer, 2>
L121: <brace, )>
L122: <brace, {>
```

```
L122: <identifer, m>
L122: <assign-op, =>
L122: <identifer, c>
L122: <arith-op, />
L122: <identifer, weight>
L122: <delimeter, ;>
L123: <identifer, c>
L123: <assign-op, =>
L123: <identifer, c>
L123: <arith-op, %>
L123: <identifer, weight>
L123: <delimeter, ;>
L124: <keyword, if>
L124: <brace, (>
L124: <identifer, m>
L124: <relop, ==>
L124: <brace, )>
L125: <identifer, p>
L125: <assign-op, =>
L125: <identifer, HT>
L125: <brace, [>
L125: <identifer, p>
L125: <brace, ]>
L125: <postfix operator, .>
L125: <delimeter, ,>
L125: <identifer, lchild>
L125: <delimeter, ;>
L126: <keyword, if>
L126: <brace, (>
L126: <identifer, m>
L126: <relop, ==>
L126: <integer, 1>
L126: <brace, )>
L127: <identifer, p>
L127: <assign-op, =>
L127: <identifer, HT>
L127: <brace, [>
L127: <identifer, p>
L127: <brace, ]>
L127: <postfix operator, .>
L127: <delimeter, ,>
L127: <identifer, rchild>
L127: <delimeter, ;>
L128: <keyword, if>
L128: <brace, (>
L128: <identifer, HT>
L128: <brace, [>
L128: <identifer, p>
L128: <brace, ]>
L128: <postfix operator, .>
L128: <delimeter, ,>
L128: <identifer, lchild>
L128: <relop, ==>
L128: <arith-op, ->
L128: <integer, 1>
L128: <logic-op, &&>
L128: <identifer, HT>
L128: <brace, [>
```

```
L128: <identifer, p>
L128: <brace, ]>
L128: <postfix operator, .>
L128: <delimeter, ,>
L128: <identifer, rchild>
L128: <relop, ==>
L128: <arith-op, ->
L128: <integer, 1>
L128: <brace, )>
L129: <brace, {>
L129: <identifer, fputc>
L129: <brace, (>
L129: <identifer, p>
L129: <delimeter, ,>
L129: <identifer, fptr2>
L129: <brace, )>
L129: <delimeter, ;>
L130: <identifer, p>
L130: <assign-op, =>
L130: <identifer, i>
L130: <delimeter, ;>
L131: <keyword, if>
L131: <brace, (>
L131: <identifer, num>
L131: <relop, ==>
L131: <arith-op, ->
L131: <integer, 1>
L131: <logic-op, &&>
L131: <identifer, weight>
L131: <relop, ==>
L131: <identifer, pos>
L131: <arith-op, *>
L131: <integer, 2>
L131: <brace, )>
L132: <keyword, break>
L132: <delimeter, ;>
L132: <brace, }>
L133: <brace, }>
L134: <brace, }>
L135: <identifer, printf>
L135: <brace, (>
L135: <string, 哈夫曼编码结束, 输出到文件>
L135: <brace, )>
L135: <delimeter, ;>
L136: <identifer, fclose>
L136: <brace, (>
L136: <identifer, fptr2>
L136: <brace, )>
L136: <delimeter, ;>
L137: <keyword, return>
L137: <delimeter, ;>
L138: <brace, }>
L140: <keyword, int>
L140: <identifer, main>
L140: <brace, (>
L140: <keyword, void>
L140: <brace, )>
L141: <brace, {>
```

```
L141: <keyword, int>
L141: <identifer, n>
L141: <delimeter, ,>
L141: <identifer, i>
L141: <delimeter, ,>
L141: <identifer, j>
L141: <delimeter, ,>
L141: <identifer, k>
L141: <delimeter, ,>
L141: <identifer, weight>
L141: <delimeter, ,>
L141: <identifer, pos>
L141: <delimeter, ,>
L141: <identifer, num>
L141: <assign-op, =>
L141: <delimeter, ;>
L142: <keyword, char>
L142: <identifer, c>
L142: <delimeter, ,>
L142: <identifer, ch>
L142: <delimeter, ;>
L143: <keyword, char>
L143: <arith-op, *>
L143: <arith-op, *>
L143: <identifer, HC>
L143: <delimeter, ;>
L144: <identifer, FILE>
L144: <arith-op, *>
L144: <identifer, fptr0>
L144: <delimeter, ;>
L145: <identifer, FILE>
L145: <arith-op, *>
L145: <identifer, fptr1>
L145: <delimeter, ;>
L146: <keyword, if>
L146: <brace, (>
L146: <brace, (>
L146: <identifer, fptr0>
L146: <assign-op, =>
L146: <identifer, fopen>
L146: <brace, (>
L146: <string, out0.txt>
L146: <delimeter, ,>
L146: <string, w+>
L146: <brace, )>
L146: <brace, )>
L146: <relop, ==>
L146: <identifer, NULL>
L146: <brace, )>
L147: <identifer, printf>
L147: <brace, (>
L147: <string, can't␣open␣the␣object␣file\n>
L147:␣<brace,␣)>
L147:␣<delimeter,␣;>
L148:␣<keyword,␣if>
L148:␣<brace,␣(>
L148:␣<brace,␣(>
L148:␣<identifer,␣fptr1>
```

```
L148:␣<assign-op,␣=>
L148:␣<identifer,␣fopen>
L148:␣<brace,␣(>
L148:␣<string,␣out1.txt>
L148:␣<delimeter,␣,>
L148:␣<string,␣wb+>
L148:␣<brace,␣)>
L148:␣<brace,␣)>
L148:␣<relop,␣==>
L148:␣<identifer,␣NULL>
L148:␣<brace,␣)>
L149:␣<identifer,␣printf>
L149:␣<brace,␣(>
L149:␣<string,␣can't open the object file\n>
L149: <brace, )>
L149: <delimeter, ;>
L150: <identifer, printf>
L150: <brace, (>
L150: <string, 对象文件无法打开>
L150: <brace, )>
L150: <delimeter, ;>
L151: <keyword, char>
L151: <identifer, filename>
L151: <brace, [>
L151: <integer, 100>
L151: <brace, ]>
L151: <delimeter, ;>
L152: <identifer, gets>
L152: <brace, (>
L152: <identifer, filename>
L152: <brace, )>
L152: <delimeter, ;>
L153: <identifer, n>
L153: <assign-op, =>
L153: <identifer, CreateHuffmantree>
L153: <brace, (>
L153: <identifer, filename>
L153: <brace, )>
L153: <delimeter, ;>
L154: <keyword, if>
L154: <brace, (>
L154: <identifer, n>
L154: <relop, ==>
L154: <brace, )>
L155: <keyword, return>
L155: <delimeter, ;>
L156: <keyword, if>
L156: <brace, (>
L156: <identifer, n>
L156: <relop, ==>
L156: <integer, 1>
L156: <brace, )>
L157: <brace, {>
L157: <identifer, rewind>
L157: <brace, (>
L157: <identifer, fptr>
L157: <brace, )>
L157: <delimeter, ;>
```

```
L158: <keyword, while>
L158: <brace, (>
L158: <brace, (>
L158: <identifer, c>
L158: <assign-op, =>
L158: <identifer, fgetc>
L158: <brace, (>
L158: <identifer, fptr>
L158: <brace, )>
L158: <brace, )>
L158: <relop, !=>
L158: <identifer, EOF>
L158: <brace, )>
L159: <brace, {>
L159: <identifer, k>
L159: <arith-op, ++>
L159: <delimeter, ;>
L160: <identifer, ch>
L160: <assign-op, =>
L160: <identifer, c>
L160: <delimeter, ;>
L161: <identifer, fputc>
L161: <brace, (>
L161: <char, 0>
L161: <delimeter, ,>
L161: <identifer, fptr0>
L161: <brace, )>
L161: <delimeter, ;>
L162: <keyword, if>
L162: <brace, (>
L162: <identifer, k>
L162: <arith-op, %>
L162: <integer, 8>
L162: <relop, ==>
L162: <brace, )>
L163: <identifer, fputc>
L163: <brace, (>
L163: <brace, (>
L163: <keyword, char>
L163: <brace, )>
L163: <delimeter, ,>
L163: <identifer, fptr1>
L163: <brace, )>
L163: <delimeter, ;>
L163: <brace, }>
L164: <identifer, fputc>
L164: <brace, (>
L164: <brace, (>
L164: <keyword, char>
L164: <brace, )>
L164: <delimeter, ,>
L164: <identifer, fptr1>
L164: <brace, )>
L164: <delimeter, ;>
L164: <brace, }>
L167: <keyword, else>
L167: <brace, {>
L167: <identifer, HC>
```

```
L167: <assign-op, =>
L167: <identifer, CreateHuffmancode>
L167: <brace, (>
L167: <identifer, n>
L167: <brace, )>
L167: <delimeter, ;>
L169: <keyword, while>
L169: <brace, (>
L169: <brace, (>
L169: <identifer, i>
L169: <assign-op, =>
L169: <identifer, fgetc>
L169: <brace, (>
L169: <identifer, fptr>
L169: <brace, )>
L169: <brace, )>
L169: <relop, !=>
L169: <identifer, EOF>
L169: <brace, )>
L170: <identifer, fputs>
L170: <brace, (>
L170: <identifer, HC>
L170: <brace, [>
L170: <identifer, i>
L170: <brace, ]>
L170: <delimeter, ,>
L170: <identifer, fptr0>
L170: <brace, )>
L170: <delimeter, ;>
L173: <identifer, rewind>
L173: <brace, (>
L173: <identifer, fptr0>
L173: <brace, )>
L173: <delimeter, ;>
L174: <identifer, weight>
L174: <assign-op, =>
L174: <integer, 128>
L174: <delimeter, ;>
L175: <identifer, j>
L175: <assign-op, =>
L175: <delimeter, ;>
L176: <keyword, while>
L176: <brace, (>
L176: <brace, (>
L176: <identifer, i>
L176: <assign-op, =>
L176: <identifer, fgetc>
L176: <brace, (>
L176: <identifer, fptr0>
L176: <brace, )>
L176: <brace, )>
L176: <relop, !=>
L176: <identifer, EOF>
L176: <brace, )>
L177: <brace, {>
L177: <identifer, j>
L177: <arith-op, +=>
L177: <brace, (>
```

```
L177: <identifer, i>
L177: <arith-op, ->
L177: <integer, 48>
L177: <brace, )>
L177: <arith-op, *>
L177: <identifer, weight>
L177: <delimeter, ;>
L178: <identifer, weight>
L178: <assign-op, =>
L178: <identifer, weight>
L178: <arith-op, />
L178: <integer, 2>
L178: <delimeter, ;>
L179: <keyword, if>
L179: <brace, (>
L179: <identifer, weight>
L179: <relop, ==>
L179: <brace, )>
L180: <brace, {>
L180: <identifer, weight>
L180: <assign-op, =>
L180: <integer, 128>
L180: <delimeter, ;>
L181: <identifer, fputc>
L181: <brace, (>
L181: <brace, (>
L181: <keyword, unsigned>
L181: <keyword, char>
L181: <brace, )>
L181: <identifer, j>
L181: <delimeter, ,>
L181: <identifer, fptr1>
L181: <brace, )>
L181: <delimeter, ;>
L182: <identifer, num>
L182: <arith-op, ++>
L182: <delimeter, ;>
L183: <identifer, j>
L183: <assign-op, =>
L183: <delimeter, ;>
L183: <brace, }>
L184: <brace, }>
L185: <keyword, if>
L185: <brace, (>
L185: <identifer, weight>
L185: <relop, !=>
L185: <brace, )>
L186: <brace, {>
L186: <identifer, fputc>
L186: <brace, (>
L186: <identifer, j>
L186: <delimeter, ,>
L186: <identifer, fptr1>
L186: <brace, )>
L186: <delimeter, ;>
L187: <identifer, pos>
L187: <assign-op, =>
L187: <identifer, weight>
```

```
L187: <delimeter, ;>
L187: <brace, }>
L187: <brace, }>
L188: <identifer, printf>
L188: <brace, (>
L188: <string, 处理过程如下:\n>
L188: <brace, )>
L188: <delimeter, ;>
L189: <identifer, c>
L189: <assign-op, =>
L189: <identifer, getchar>
L189: <brace, (>
L189: <brace, )>
L189: <delimeter, ;>
L190: <keyword, if>
L190: <brace, (>
L190: <identifer, c>
L190: <relop, ==>
L190: <char, y>
L190: <brace, )>
L191: <keyword, if>
L191: <brace, (>
L191: <identifer, n>
L191: <relop, >>
L191: <integer, 1>
L191: <brace, )>
L192: <brace, {>
L193: <identifer, Huffmandecoding>
L193: <brace, (>
L193: <identifer, fptr1>
L193: <delimeter, ,>
L193: <identifer, num>
L193: <delimeter, ,>
L193: <identifer, pos>
L193: <delimeter, ,>
L193: <identifer, filename>
L193: <delimeter, ,>
L193: <identifer, filetype>
L193: <brace, (>
L193: <identifer, filename>
L193: <brace, )>
L193: <brace, )>
L193: <delimeter, ;>
L194: <identifer, free>
L194: <brace, (>
L194: <identifer, HT>
L194: <brace, )>
L194: <delimeter, ;>
L195: <keyword, for>
L195: <brace, (>
L195: <identifer, i>
L195: <assign-op, =>
L195: <delimeter, ;>
L195: <identifer, i>
L195: <relop, <>
L195: <integer, 256>
L195: <delimeter, ;>
L195: <identifer, i>
```

```
L195: <arith-op, ++>
L195: <brace, )>
L196: <identifer, free>
L196: <brace, (>
L196: <identifer, HC>
L196: <brace, [>
L196: <identifer, i>
L196: <brace, ]>
L196: <brace, )>
L196: <delimeter, ;>
L197: <identifer, free>
L197: <brace, (>
L197: <identifer, HC>
L197: <brace, )>
L197: <delimeter, ;>
L197: <brace, }>
L199: <keyword, else>
L199: <brace, {>
L199: <identifer, rewind>
L199: <brace, (>
L199: <identifer, fptr1>
L199: <brace, )>
L199: <delimeter, ;>
L200: <identifer, FILE>
L200: <arith-op, *>
L200: <identifer, fptr2>
L200: <delimeter, ;>
L201: <keyword, char>
L201: <identifer, objectfile>
L201: <brace, [>
L201: <integer, 10>
L201: <brace, ]>
L201: <assign-op, =>
L201: <string, out2>
L201: <delimeter, ;>
L202: <identifer, strcat>
L202: <brace, (>
L202: <identifer, objectfile>
L202: <delimeter, ,>
L202: <bitop, &>
L202: <identifer, filename>
L202: <brace, [>
L202: <identifer, filetype>
L202: <brace, (>
L202: <identifer, filename>
L202: <brace, )>
L202: <brace, ]>
L202: <brace, )>
L202: <delimeter, ;>
L202: <delimeter, ;>
L203: <keyword, if>
L203: <brace, (>
L203: <brace, (>
L203: <identifer, fptr2>
L203: <assign-op, =>
L203: <identifer, fopen>
L203: <brace, (>
L203: <identifer, objectfile>
```

```
L203: <delimeter , ,>
L203: <string , wb>
L203: <brace , )>
L203: <brace , )>
L203: <relop , ==>
L203: <identifer , NULL>
L203: <brace , )>
L204: <identifer , printf>
L204: <brace , (>
L204: <string , can't␣open␣the␣object␣file\n>
L204:␣<brace ,␣)>
L204:␣<delimeter ,␣;>
L205:␣<keyword ,␣for>
L205:␣<brace ,␣(>
L205:␣<identifer ,␣i>
L205:␣<assign-op ,␣=>
L205:␣<delimeter ,␣;>
L205:␣<identifer ,␣i>
L205:␣<relop ,␣<>
L205:␣<identifer ,␣k>
L205:␣<delimeter ,␣;>
L205:␣<identifer ,␣i>
L205:␣<arith-op ,␣++>
L205:␣<brace ,␣)>
L206:␣<identifer ,␣fputc>
L206:␣<brace ,␣(>
L206:␣<identifer ,␣ch>
L206:␣<delimeter ,␣,>
L206:␣<identifer ,␣fptr2>
L206:␣<brace ,␣)>
L206:␣<delimeter ,␣;>
L207:␣<identifer ,␣printf>
L207:␣<brace ,␣(>
L207:␣<string ,␣处理结果写入到out2中>
L207:␣<brace ,␣)>
L207:␣<delimeter ,␣;>
L208:␣<identifer ,␣fclose>
L208:␣<brace ,␣(>
L208:␣<identifer ,␣fptr2>
L208:␣<brace ,␣)>
L208:␣<delimeter ,␣;>
L208:␣<brace ,␣}>
L211:␣<identifer ,␣fclose>
L211:␣<brace ,␣(>
L211:␣<identifer ,␣fptr>
L211:␣<brace ,␣)>
L211:␣<delimeter ,␣;>
L212:␣<identifer ,␣fclose>
L212:␣<brace ,␣(>
L212:␣<identifer ,␣fptr0>
L212:␣<brace ,␣)>
L212:␣<delimeter ,␣;>
L213:␣<identifer ,␣fclose>
L213:␣<brace ,␣(>
L213:␣<identifer ,␣fptr1>
L213:␣<brace ,␣)>
L213:␣<delimeter ,␣;>
L214:␣<keyword ,␣return>
```

```
L214: <delimeter , ;>
L215: <brace , }>

====== Statistic information ======
215 lines , 4855 characters , 119 symbols
<keyword , typedef> appeared 1 times
<keyword , struct> appeared 1 times
<identifer , huf> appeared 1 times
<brace , {> appeared 31 times
<keyword , int> appeared 20 times
<identifer , weight> appeared 35 times
<delimeter , ;> appeared 148 times
<identifer , parent> appeared 11 times
<delimeter , ,> appeared 86 times
<identifer , lchild> appeared 6 times
<identifer , rchild> appeared 5 times
<brace , }> appeared 31 times
<identifer , HTNode> appeared 2 times
<arith -op , *> appeared 39 times
<identifer , Huffmantree> appeared 3 times
<identifer , FILE> appeared 6 times
<identifer , fptr> appeared 8 times
<assign -op , => appeared 76 times
<identifer , NULL> appeared 7 times
<identifer , HT> appeared 39 times
<identifer , filetype> appeared 3 times
<brace , (> appeared 127 times
<keyword , char> appeared 18 times
<identifer , ch> appeared 6 times
<brace , [> appeared 55 times
<brace , ]> appeared 55 times
<brace , )> appeared 127 times
<identifer , i> appeared 79 times
<identifer , n> appeared 22 times
<keyword , while> appeared 7 times
<relop , !=> appeared 8 times
<char , \0> appeared 3 times
<keyword , if> appeared 27 times
<relop , ==> appeared 24 times
<char , .> appeared 1 times
<arith -op , ++> appeared 15 times
<keyword , return> appeared 8 times
<keyword , void> appeared 3 times
<identifer , Select> appeared 2 times
<identifer , s1> appeared 13 times
<identifer , s2> appeared 13 times
<identifer , temp> appeared 3 times
<keyword , for> appeared 11 times
<relop , <=> appeared 6 times
<postfix operator , .> appeared 36 times
<relop , >> appeared 6 times
<logic -op , &&> appeared 6 times
<arith -op , ->> appeared 15 times
<integer , 1> appeared 20 times
<keyword , break> appeared 4 times
<arith -op , +> appeared 5 times
<relop , <> appeared 7 times
<keyword , else> appeared 5 times
```

```
<identifer, CreateHuffmantree> appeared 2 times
<identifer, filename> appeared 11 times
<identifer, k> appeared 10 times
<identifer, malloc> appeared 3 times
<keyword, sizeof> appeared 3 times
<integer, 512> appeared 2 times
<identifer, fopen> appeared 5 times
<string, rb> appeared 1 times
<identifer, printf> appeared 9 times
<string, can't open the source file\n> appeared 1 times
<integer, 255> appeared 2 times
<identifer, fgetc> appeared 5 times
<identifer, EOF> appeared 5 times
<identifer, rewind> appeared 5 times
<integer, 256> appeared 5 times
<bitop, &> appeared 5 times
<identifer, CreateHuffmancode> appeared 2 times
<identifer, j> appeared 13 times
<identifer, start> appeared 6 times
<identifer, pre> appeared 6 times
<identifer, HC> appeared 10 times
<identifer, cd> appeared 5 times
<arith-op, --> appeared 2 times
<char, 0> appeared 2 times
<char, 1> appeared 1 times
<identifer, strcpy> appeared 1 times
<identifer, Huffmandecoding> appeared 2 times
<identifer, fptr1> appeared 12 times
<identifer, num> appeared 6 times
<identifer, pos> appeared 5 times
<identifer, fptr2> appeared 8 times
<identifer, objectfile> appeared 6 times
<integer, 10> appeared 2 times
<string, out2> appeared 2 times
<identifer, strcat> appeared 2 times
<string, wb> appeared 2 times
<string, can't open the object file\n> appeared 4 times
<identifer, c> appeared 10 times
<identifer, m> appeared 4 times
<identifer, p> appeared 10 times
<integer, 128> appeared 3 times
<arith-op, /> appeared 3 times
<integer, 2> appeared 3 times
<arith-op, %> appeared 2 times
<identifer, fputc> appeared 7 times
<string, 哈夫曼编码结束, 输出到文件> appeared 1 times
<identifer, fclose> appeared 5 times
<identifer, main> appeared 1 times
<identifer, fptr0> appeared 7 times
<string, out0.txt> appeared 1 times
<string, w+> appeared 1 times
<string, out1.txt> appeared 1 times
<string, wb+> appeared 1 times
<string, 对象文件无法打开> appeared 1 times
<integer, 100> appeared 1 times
<identifer, gets> appeared 1 times
<integer, 8> appeared 1 times
<identifer, fputs> appeared 1 times
```

```
<arith-op, +=> appeared 1 times
<integer, 48> appeared 1 times
<keyword, unsigned> appeared 1 times
<string, 处理过程如下:\n> appeared 1 times
<identifer, getchar> appeared 1 times
<char, y> appeared 1 times
<identifer, free> appeared 3 times
<string, 处理结果写入到out2中> appeared 1 times
======End of statistic information======
```

### 7.3.2 FLEX 版词法分析程序测试

```
L1:     <keyword, typedef>
L1:     <keyword, struct>
L1:     <identifier, huf>
L2:     <brace, {>
L2:     <keyword, int>
L2:     <identifier, weight>
L2:     <delimeter, ;>
L3:     <keyword, int>
L3:     <identifier, parent>
L3:     <delimeter, ,>
L3:     <identifier, lchild>
L3:     <delimeter, ,>
L3:     <identifier, rchild>
L3:     <delimeter, ;>
L4:     <brace, }>
L4:     <identifier, HTNode>
L4:     <delimeter, ,>
L4:     <arith-op, *>
L4:     <identifier, Huffmantree>
L4:     <delimeter, ;>
L6:     <identifier, FILE>
L6:     <arith-op, *>
L6:     <identifier, fptr>
L6:     <assign-op, =>
L6:     <identifier, NULL>
L6:     <delimeter, ;>
L8:     <identifier, Huffmantree>
L8:     <identifier, HT>
L8:     <assign-op, =>
L8:     <identifier, NULL>
L8:     <delimeter, ;>
L11:    <keyword, int>
L11:    <identifier, filetype>
L11:    <brace, (>
L11:    <keyword, char>
L11:    <identifier, ch>
L11:    <brace, [>
L11:    <brace, ]>
L11:    <brace, )>
L12:    <brace, {>
L12:    <keyword, int>
L12:    <identifier, i>
L12:    <assign-op, =>
L12:    <integer, 0>
```

```
L12:    <delimeter, ,>
L12:    <identifier, n>
L12:    <delimeter, ;>
L13:    <keyword, while>
L13:    <brace, (>
L13:    <identifier, ch>
L13:    <brace, [>
L13:    <identifier, i>
L13:    <brace, ]>
L13:    <relop, !=>
L13:    <char, '\0'>
L13:    <brace, )>
L14:    <brace, {>
L14:    <keyword, if>
L14:    <brace, (>
L14:    <identifier, ch>
L14:    <brace, [>
L14:    <identifier, i>
L14:    <brace, ]>
L14:    <relop, ==>
L14:    <char, '.'>
L14:    <brace, )>
L15:    <identifier, n>
L15:    <assign-op, =>
L15:    <identifier, i>
L15:    <delimeter, ;>
L16:    <identifier, i>
L16:    <arith-op, ++>
L16:    <delimeter, ;>
L16:    <brace, }>
L17:    <keyword, return>
L17:    <identifier, n>
L17:    <delimeter, ;>
L17:    <brace, }>
L20:    <keyword, void>
L20:    <identifier, Select>
L20:    <brace, (>
L20:    <keyword, int>
L20:    <identifier, n>
L20:    <delimeter, ,>
L20:    <keyword, int>
L20:    <arith-op, *>
L20:    <identifier, s1>
L20:    <delimeter, ,>
L20:    <keyword, int>
L20:    <arith-op, *>
L20:    <identifier, s2>
L20:    <brace, )>
L21:    <brace, {>
L21:    <keyword, int>
L21:    <identifier, i>
L21:    <delimeter, ,>
L21:    <identifier, temp>
L21:    <delimeter, ;>
L22:    <keyword, for>
L22:    <brace, (>
L22:    <identifier, i>
L22:    <assign-op, =>
```

```
L22:    <integer, 0>
L22:    <delimeter, ;>
L22:    <identifier, i>
L22:    <relop, <=>
L22:    <identifier, n>
L22:    <delimeter, ;>
L22:    <identifier, i>
L22:    <arith-op, ++>
L22:    <brace, )>
L23:    <keyword, if>
L23:    <brace, (>
L23:    <identifier, HT>
L23:    <brace, [>
L23:    <identifier, i>
L23:    <brace, ]>
L23:    <postfix operator, .>
L23:    <identifier, weight>
L23:    <relop, >>
L23:    <integer, 0>
L23:    <logic-op, &&>
L23:    <identifier, HT>
L23:    <brace, [>
L23:    <identifier, i>
L23:    <brace, ]>
L23:    <postfix operator, .>
L23:    <identifier, parent>
L23:    <relop, ==>
L23:    <arith-op, ->
L23:    <integer, 1>
L23:    <brace, )>
L24:    <brace, {>
L24:    <arith-op, *>
L24:    <identifier, s1>
L24:    <assign-op, =>
L24:    <identifier, i>
L24:    <delimeter, ;>
L25:    <keyword, break>
L25:    <delimeter, ;>
L25:    <brace, }>
L26:    <keyword, for>
L26:    <brace, (>
L26:    <identifier, i>
L26:    <assign-op, =>
L26:    <identifier, i>
L26:    <arith-op, +>
L26:    <integer, 1>
L26:    <delimeter, ;>
L26:    <identifier, i>
L26:    <relop, <=>
L26:    <identifier, n>
L26:    <delimeter, ;>
L26:    <identifier, i>
L26:    <arith-op, ++>
L26:    <brace, )>
L27:    <brace, {>
L27:    <keyword, if>
L27:    <brace, (>
L27:    <identifier, HT>
```

```
L27:    <brace, [>
L27:    <identifier, i>
L27:    <brace, ]>
L27:    <postfix operator, .>
L27:    <identifier, weight>
L27:    <relop, >>
L27:    <integer, 0>
L27:    <logic-op, &&>
L27:    <identifier, HT>
L27:    <brace, [>
L27:    <identifier, i>
L27:    <brace, ]>
L27:    <postfix operator, .>
L27:    <identifier, parent>
L27:    <relop, ==>
L27:    <arith-op, ->
L27:    <integer, 1>
L27:    <brace, )>
L28:    <brace, {>
L28:    <arith-op, *>
L28:    <identifier, s2>
L28:    <assign-op, =>
L28:    <identifier, i>
L28:    <delimeter, ;>
L29:    <keyword, break>
L29:    <delimeter, ;>
L29:    <brace, }>
L30:    <brace, }>
L31:    <keyword, if>
L31:    <brace, (>
L31:    <identifier, HT>
L31:    <brace, [>
L31:    <arith-op, *>
L31:    <identifier, s2>
L31:    <brace, ]>
L31:    <postfix operator, .>
L31:    <identifier, weight>
L31:    <relop, <>
L31:    <identifier, HT>
L31:    <brace, [>
L31:    <arith-op, *>
L31:    <identifier, s1>
L31:    <brace, ]>
L31:    <postfix operator, .>
L31:    <identifier, weight>
L31:    <brace, )>
L32:    <brace, {>
L32:    <identifier, temp>
L32:    <assign-op, =>
L32:    <arith-op, *>
L32:    <identifier, s1>
L32:    <delimeter, ;>
L33:    <arith-op, *>
L33:    <identifier, s1>
L33:    <assign-op, =>
L33:    <arith-op, *>
L33:    <identifier, s2>
L33:    <delimeter, ;>
```

```
L34:    <arith-op, *>
L34:    <identifier, s2>
L34:    <assign-op, =>
L34:    <identifier, temp>
L34:    <delimeter, ;>
L35:    <brace, }>
L36:    <keyword, for>
L36:    <brace, (>
L36:    <identifier, i>
L36:    <assign-op, =>
L36:    <identifier, i>
L36:    <arith-op, +>
L36:    <integer, 1>
L36:    <delimeter, ;>
L36:    <identifier, i>
L36:    <relop, <=>
L36:    <identifier, n>
L36:    <delimeter, ;>
L36:    <identifier, i>
L36:    <arith-op, ++>
L36:    <brace, )>
L37:    <keyword, if>
L37:    <brace, (>
L37:    <identifier, HT>
L37:    <brace, [>
L37:    <identifier, i>
L37:    <brace, ]>
L37:    <postfix operator, .>
L37:    <identifier, weight>
L37:    <relop, >>
L37:    <integer, 0>
L37:    <logic-op, &&>
L37:    <identifier, HT>
L37:    <brace, [>
L37:    <identifier, i>
L37:    <brace, ]>
L37:    <postfix operator, .>
L37:    <identifier, parent>
L37:    <relop, ==>
L37:    <arith-op, ->
L37:    <integer, 1>
L37:    <brace, )>
L38:    <keyword, if>
L38:    <brace, (>
L38:    <identifier, HT>
L38:    <brace, [>
L38:    <identifier, i>
L38:    <brace, ]>
L38:    <postfix operator, .>
L38:    <identifier, weight>
L38:    <relop, <>
L38:    <identifier, HT>
L38:    <brace, [>
L38:    <arith-op, *>
L38:    <identifier, s1>
L38:    <brace, ]>
L38:    <postfix operator, .>
L38:    <identifier, weight>
```

```
L38:    <brace, )>
L39:    <brace, {>
L39:    <arith-op, *>
L39:    <identifier, s2>
L39:    <assign-op, =>
L39:    <arith-op, *>
L39:    <identifier, s1>
L39:    <delimeter, ;>
L40:    <arith-op, *>
L40:    <identifier, s1>
L40:    <assign-op, =>
L40:    <identifier, i>
L40:    <delimeter, ;>
L40:    <brace, }>
L41:    <keyword, else>
L41:    <keyword, if>
L41:    <brace, (>
L41:    <identifier, HT>
L41:    <brace, [>
L41:    <identifier, i>
L41:    <brace, ]>
L41:    <postfix operator, .>
L41:    <identifier, weight>
L41:    <relop, <>
L41:    <identifier, HT>
L41:    <brace, [>
L41:    <arith-op, *>
L41:    <identifier, s2>
L41:    <brace, ]>
L41:    <postfix operator, .>
L41:    <identifier, weight>
L41:    <brace, )>
L42:    <arith-op, *>
L42:    <identifier, s2>
L42:    <assign-op, =>
L42:    <identifier, i>
L42:    <delimeter, ;>
L43:    <keyword, return>
L43:    <delimeter, ;>
L44:    <brace, }>
L47:    <keyword, int>
L47:    <identifier, CreateHuffmantree>
L47:    <brace, (>
L47:    <keyword, char>
L47:    <identifier, filename>
L47:    <brace, [>
L47:    <brace, ]>
L47:    <brace, )>
L48:    <brace, {>
L48:    <keyword, int>
L48:    <identifier, i>
L48:    <delimeter, ,>
L48:    <identifier, k>
L48:    <delimeter, ,>
L48:    <identifier, n>
L48:    <assign-op, =>
L48:    <integer, 0>
L48:    <delimeter, ;>
```

```
L49:    <keyword, int>
L49:    <identifier, s1>
L49:    <delimeter, ,>
L49:    <identifier, s2>
L49:    <delimeter, ;>
L50:    <identifier, HT>
L50:    <assign-op, =>
L50:    <brace, (>
L50:    <identifier, Huffmantree>
L50:    <brace, )>
L50:    <identifier, malloc>
L50:    <brace, (>
L50:    <keyword, sizeof>
L50:    <brace, (>
L50:    <identifier, HTNode>
L50:    <brace, )>
L50:    <arith-op, *>
L50:    <integer, 512>
L50:    <brace, )>
L50:    <delimeter, ;>
L51:    <keyword, if>
L51:    <brace, (>
L51:    <brace, (>
L51:    <identifier, fptr>
L51:    <assign-op, =>
L51:    <identifier, fopen>
L51:    <brace, (>
L51:    <identifier, filename>
L51:    <delimeter, ,>
L51:    <string, "rb">
L51:    <brace, )>
L51:    <brace, )>
L51:    <relop, ==>
L51:    <identifier, NULL>
L51:    <brace, )>
L52:    <brace, {>
L52:    <identifier, printf>
L52:    <brace, (>
L52:    <string, "can't␣open␣the␣source␣file\n">
L52:    <brace, )>
L52:    <delimeter, ;>
L53:    <keyword, return>
L53:    <integer, 0>
L53:    <delimeter, ;>
L53:    <brace, }>
L54:    <keyword, for>
L54:    <brace, (>
L54:    <identifier, i>
L54:    <assign-op, =>
L54:    <integer, 0>
L54:    <delimeter, ;>
L54:    <identifier, i>
L54:    <relop, <=>
L54:    <integer, 255>
L54:    <delimeter, ;>
L54:    <identifier, i>
L54:    <arith-op, ++>
L54:    <brace, )>
```

```
L55:    <brace, {>
L55:    <identifier, HT>
L55:    <brace, [>
L55:    <identifier, i>
L55:    <brace, ]>
L55:    <postfix operator, .>
L55:    <identifier, parent>
L55:    <assign-op, =>
L55:    <identifier, HT>
L55:    <brace, [>
L55:    <identifier, i>
L55:    <brace, ]>
L55:    <postfix operator, .>
L55:    <identifier, lchild>
L55:    <assign-op, =>
L55:    <identifier, HT>
L55:    <brace, [>
L55:    <identifier, i>
L55:    <brace, ]>
L55:    <postfix operator, .>
L55:    <identifier, rchild>
L55:    <assign-op, =>
L55:    <arith-op, ->
L55:    <integer, 1>
L55:    <delimeter, ;>
L56:    <identifier, HT>
L56:    <brace, [>
L56:    <identifier, i>
L56:    <brace, ]>
L56:    <postfix operator, .>
L56:    <identifier, weight>
L56:    <assign-op, =>
L56:    <integer, 0>
L56:    <delimeter, ;>
L56:    <brace, }>
L57:    <keyword, while>
L57:    <brace, (>
L57:    <brace, (>
L57:    <identifier, k>
L57:    <assign-op, =>
L57:    <identifier, fgetc>
L57:    <brace, (>
L57:    <identifier, fptr>
L57:    <brace, )>
L57:    <brace, )>
L57:    <relop, !=>
L57:    <identifier, EOF>
L57:    <brace, )>
L58:    <brace, {>
L58:    <keyword, if>
L58:    <brace, (>
L58:    <identifier, HT>
L58:    <brace, [>
L58:    <identifier, k>
L58:    <brace, ]>
L58:    <postfix operator, .>
L58:    <identifier, weight>
L58:    <relop, ==>
```

```
L58:    <integer, 0>
L58:    <brace, )>
L59:    <identifier, n>
L59:    <arith-op, ++>
L59:    <delimeter, ;>
L60:    <identifier, HT>
L60:    <brace, [>
L60:    <identifier, k>
L60:    <brace, ]>
L60:    <postfix operator, .>
L60:    <identifier, weight>
L60:    <arith-op, ++>
L60:    <delimeter, ;>
L61:    <brace, }>
L62:    <identifier, rewind>
L62:    <brace, (>
L62:    <identifier, fptr>
L62:    <brace, )>
L62:    <delimeter, ;>
L63:    <keyword, for>
L63:    <brace, (>
L63:    <identifier, i>
L63:    <assign-op, =>
L63:    <integer, 256>
L63:    <delimeter, ;>
L63:    <identifier, i>
L63:    <relop, <>
L63:    <integer, 256>
L63:    <arith-op, +>
L63:    <identifier, n>
L63:    <arith-op, ->
L63:    <integer, 1>
L63:    <delimeter, ;>
L63:    <identifier, i>
L63:    <arith-op, ++>
L63:    <brace, )>
L64:    <brace, {>
L64:    <identifier, Select>
L64:    <brace, (>
L64:    <identifier, i>
L64:    <arith-op, ->
L64:    <integer, 1>
L64:    <delimeter, ,>
L64:    <bitop, &>
L64:    <identifier, s1>
L64:    <delimeter, ,>
L64:    <bitop, &>
L64:    <identifier, s2>
L64:    <brace, )>
L64:    <delimeter, ;>
L65:    <identifier, HT>
L65:    <brace, [>
L65:    <identifier, s1>
L65:    <brace, ]>
L65:    <postfix operator, .>
L65:    <identifier, parent>
L65:    <assign-op, =>
L65:    <identifier, i>
```

```
L65:    <delimeter, ;>
L66:    <identifier, HT>
L66:    <brace, [>
L66:    <identifier, s2>
L66:    <brace, ]>
L66:    <postfix operator, .>
L66:    <identifier, parent>
L66:    <assign-op, =>
L66:    <identifier, i>
L66:    <delimeter, ;>
L67:    <identifier, HT>
L67:    <brace, [>
L67:    <identifier, i>
L67:    <brace, ]>
L67:    <postfix operator, .>
L67:    <identifier, lchild>
L67:    <assign-op, =>
L67:    <identifier, s1>
L67:    <delimeter, ;>
L68:    <identifier, HT>
L68:    <brace, [>
L68:    <identifier, i>
L68:    <brace, ]>
L68:    <postfix operator, .>
L68:    <identifier, rchild>
L68:    <assign-op, =>
L68:    <identifier, s2>
L68:    <delimeter, ;>
L69:    <identifier, HT>
L69:    <brace, [>
L69:    <identifier, i>
L69:    <brace, ]>
L69:    <postfix operator, .>
L69:    <identifier, weight>
L69:    <assign-op, =>
L69:    <identifier, HT>
L69:    <brace, [>
L69:    <identifier, s1>
L69:    <brace, ]>
L69:    <postfix operator, .>
L69:    <identifier, weight>
L69:    <arith-op, +>
L69:    <identifier, HT>
L69:    <brace, [>
L69:    <identifier, s2>
L69:    <brace, ]>
L69:    <postfix operator, .>
L69:    <identifier, weight>
L69:    <delimeter, ;>
L70:    <identifier, HT>
L70:    <brace, [>
L70:    <identifier, i>
L70:    <brace, ]>
L70:    <postfix operator, .>
L70:    <identifier, parent>
L70:    <assign-op, =>
L70:    <arith-op, ->
L70:    <integer, 1>
```

```
L70:    <delimeter, ;>
L71:    <brace, }>
L72:    <keyword, return>
L72:    <identifier, n>
L72:    <delimeter, ;>
L73:    <brace, }>
L76:    <keyword, char>
L76:    <arith-op, *>
L76:    <arith-op, *>
L76:    <identifier, CreateHuffmancode>
L76:    <brace, (>
L76:    <keyword, int>
L76:    <identifier, n>
L76:    <brace, )>
L77:    <brace, {>
L77:    <keyword, int>
L77:    <identifier, i>
L77:    <delimeter, ,>
L77:    <identifier, j>
L77:    <delimeter, ,>
L77:    <identifier, start>
L77:    <delimeter, ,>
L77:    <identifier, pre>
L77:    <delimeter, ;>
L78:    <keyword, char>
L78:    <arith-op, *>
L78:    <arith-op, *>
L78:    <identifier, HC>
L78:    <assign-op, =>
L78:    <brace, (>
L78:    <keyword, char>
L78:    <arith-op, *>
L78:    <arith-op, *>
L78:    <brace, )>
L78:    <identifier, malloc>
L78:    <brace, (>
L78:    <keyword, sizeof>
L78:    <brace, (>
L78:    <keyword, char>
L78:    <arith-op, *>
L78:    <brace, )>
L78:    <arith-op, *>
L78:    <integer, 256>
L78:    <brace, )>
L78:    <delimeter, ;>
L79:    <keyword, for>
L79:    <brace, (>
L79:    <identifier, i>
L79:    <assign-op, =>
L79:    <integer, 0>
L79:    <delimeter, ;>
L79:    <identifier, i>
L79:    <relop, <>
L79:    <integer, 256>
L79:    <delimeter, ;>
L79:    <identifier, i>
L79:    <arith-op, ++>
L79:    <brace, )>
```

```
L80:    <identifier, HC>
L80:    <brace, [>
L80:    <identifier, i>
L80:    <brace, ]>
L80:    <assign-op, =>
L80:    <brace, (>
L80:    <keyword, char>
L80:    <arith-op, *>
L80:    <brace, )>
L80:    <identifier, malloc>
L80:    <brace, (>
L80:    <keyword, sizeof>
L80:    <brace, (>
L80:    <keyword, char>
L80:    <brace, )>
L80:    <arith-op, *>
L80:    <brace, (>
L80:    <identifier, n>
L80:    <arith-op, +>
L80:    <integer, 1>
L80:    <brace, )>
L80:    <brace, )>
L80:    <delimeter, ;>
L81:    <keyword, char>
L81:    <identifier, cd>
L81:    <brace, [>
L81:    <identifier, n>
L81:    <brace, ]>
L81:    <delimeter, ;>
L82:    <identifier, cd>
L82:    <brace, [>
L82:    <identifier, n>
L82:    <arith-op, ->
L82:    <integer, 1>
L82:    <brace, ]>
L82:    <assign-op, =>
L82:    <char, '\0'>
L82:    <delimeter, ;>
L84:    <keyword, for>
L84:    <brace, (>
L84:    <identifier, i>
L84:    <assign-op, =>
L84:    <integer, 0>
L84:    <delimeter, ;>
L84:    <identifier, i>
L84:    <relop, <=>
L84:    <integer, 255>
L84:    <delimeter, ;>
L84:    <identifier, i>
L84:    <arith-op, ++>
L84:    <brace, )>
L85:    <brace, {>
L85:    <keyword, if>
L85:    <brace, (>
L85:    <identifier, HT>
L85:    <brace, [>
L85:    <identifier, i>
L85:    <brace, ]>
```

```
L85:    <postfix operator, .>
L85:    <identifier, weight>
L85:    <relop, ==>
L85:    <integer, 0>
L85:    <brace, )>
L86:    <identifier, HC>
L86:    <brace, [>
L86:    <identifier, i>
L86:    <brace, ]>
L86:    <assign-op, =>
L86:    <char, '\0'>
L86:    <delimeter, ;>
L87:    <keyword, else>
L88:    <brace, {>
L88:    <identifier, start>
L88:    <assign-op, =>
L88:    <identifier, n>
L88:    <arith-op, ->
L88:    <integer, 1>
L88:    <delimeter, ;>
L89:    <identifier, j>
L89:    <assign-op, =>
L89:    <identifier, i>
L89:    <delimeter, ;>
L90:    <identifier, pre>
L90:    <assign-op, =>
L90:    <identifier, HT>
L90:    <brace, [>
L90:    <identifier, j>
L90:    <brace, ]>
L90:    <postfix operator, .>
L90:    <identifier, parent>
L90:    <delimeter, ;>
L91:    <keyword, while>
L91:    <brace, (>
L91:    <identifier, pre>
L91:    <relop, !=>
L91:    <arith-op, ->
L91:    <integer, 1>
L91:    <brace, )>
L92:    <brace, {>
L92:    <identifier, start>
L92:    <arith-op, -->
L92:    <delimeter, ;>
L93:    <keyword, if>
L93:    <brace, (>
L93:    <identifier, HT>
L93:    <brace, [>
L93:    <identifier, pre>
L93:    <brace, ]>
L93:    <postfix operator, .>
L93:    <identifier, lchild>
L93:    <relop, ==>
L93:    <identifier, j>
L93:    <brace, )>
L94:    <identifier, cd>
L94:    <brace, [>
L94:    <identifier, start>
```

```
L94:    <brace, ]>
L94:    <assign-op, =>
L94:    <char, '0'>
L94:    <delimeter, ;>
L95:    <keyword, else>
L96:    <identifier, cd>
L96:    <brace, [>
L96:    <identifier, start>
L96:    <brace, ]>
L96:    <assign-op, =>
L96:    <char, '1'>
L96:    <delimeter, ;>
L97:    <identifier, j>
L97:    <assign-op, =>
L97:    <identifier, pre>
L97:    <delimeter, ;>
L98:    <identifier, pre>
L98:    <assign-op, =>
L98:    <identifier, HT>
L98:    <brace, [>
L98:    <identifier, j>
L98:    <brace, ]>
L98:    <postfix operator, .>
L98:    <identifier, parent>
L98:    <delimeter, ;>
L99:    <brace, }>
L100:   <identifier, strcpy>
L100:   <brace, (>
L100:   <identifier, HC>
L100:   <brace, [>
L100:   <identifier, i>
L100:   <brace, ]>
L100:   <delimeter, ,>
L100:   <bitop, &>
L100:   <identifier, cd>
L100:   <brace, [>
L100:   <identifier, start>
L100:   <brace, ]>
L100:   <brace, )>
L100:   <delimeter, ;>
L101:   <brace, }>
L102:   <brace, }>
L103:   <keyword, return>
L103:   <identifier, HC>
L103:   <delimeter, ;>
L104:   <brace, }>
L106:   <keyword, void>
L106:   <identifier, Huffmandecoding>
L106:   <brace, (>
L106:   <identifier, FILE>
L106:   <arith-op, *>
L106:   <identifier, fptr1>
L106:   <delimeter, ,>
L106:   <keyword, int>
L106:   <identifier, num>
L106:   <delimeter, ,>
L106:   <keyword, int>
L106:   <identifier, pos>
```

```
L106:   <delimeter , ,>
L106:   <keyword , char>
L106:   <arith-op , *>
L106:   <identifier , filename>
L106:   <delimeter , ,>
L106:   <keyword , int>
L106:   <identifier , k>
L106:   <brace , )>
L107:   <brace , {>
L107:   <identifier , rewind>
L107:   <brace , (>
L107:   <identifier , fptr1>
L107:   <brace , )>
L107:   <delimeter , ;>
L108:   <identifier , FILE>
L108:   <arith-op , *>
L108:   <identifier , fptr2>
L108:   <delimeter , ;>
L109:   <keyword , char>
L109:   <identifier , objectfile>
L109:   <brace , [>
L109:   <integer , 10>
L109:   <brace , ]>
L109:   <assign-op , =>
L109:   <string , "out2">
L109:   <delimeter , ;>
L110:   <identifier , strcat>
L110:   <brace , (>
L110:   <identifier , objectfile>
L110:   <delimeter , ,>
L110:   <bitop , &>
L110:   <identifier , filename>
L110:   <brace , [>
L110:   <identifier , k>
L110:   <brace , ]>
L110:   <brace , )>
L110:   <delimeter , ;>
L110:   <delimeter , ;>
L111:   <keyword , if>
L111:   <brace , (>
L111:   <brace , (>
L111:   <identifier , fptr2>
L111:   <assign-op , =>
L111:   <identifier , fopen>
L111:   <brace , (>
L111:   <identifier , objectfile>
L111:   <delimeter , ,>
L111:   <string , "wb">
L111:   <brace , )>
L111:   <brace , )>
L111:   <relop , ==>
L111:   <identifier , NULL>
L111:   <brace , )>
L112:   <identifier , printf>
L112:   <brace , (>
L112:   <string , "can't␣open␣the␣object␣file\n">
L112:   <brace , )>
L112:   <delimeter , ;>
```

```
L113:   <keyword, int>
L113:   <identifier, c>
L113:   <delimeter, ;>
L114:   <keyword, int>
L114:   <identifier, i>
L114:   <delimeter, ,>
L114:   <identifier, j>
L114:   <delimeter, ,>
L114:   <identifier, m>
L114:   <delimeter, ,>
L114:   <identifier, p>
L114:   <delimeter, ,>
L114:   <identifier, weight>
L114:   <delimeter, ;>
L115:   <keyword, for>
L115:   <brace, (>
L115:   <identifier, i>
L115:   <assign-op, =>
L115:   <integer, 0>
L115:   <delimeter, ;>
L115:   <identifier, i>
L115:   <relop, <=>
L115:   <integer, 512>
L115:   <delimeter, ;>
L115:   <identifier, i>
L115:   <arith-op, ++>
L115:   <brace, )>
L116:   <keyword, if>
L116:   <brace, (>
L116:   <brace, (>
L116:   <identifier, HT>
L116:   <brace, [>
L116:   <identifier, i>
L116:   <brace, ]>
L116:   <postfix operator, .>
L116:   <identifier, parent>
L116:   <relop, ==>
L116:   <arith-op, ->
L116:   <integer, 1>
L116:   <brace, )>
L116:   <logic-op, &&>
L116:   <identifier, HT>
L116:   <brace, [>
L116:   <identifier, i>
L116:   <brace, ]>
L116:   <postfix operator, .>
L116:   <identifier, weight>
L116:   <relop, >>
L116:   <integer, 0>
L116:   <brace, )>
L117:   <keyword, break>
L117:   <delimeter, ;>
L118:   <identifier, p>
L118:   <assign-op, =>
L118:   <identifier, i>
L118:   <delimeter, ;>
L119:   <keyword, while>
L119:   <brace, (>
```

```
L119:   <brace, (>
L119:   <identifier, c>
L119:   <assign-op, =>
L119:   <identifier, fgetc>
L119:   <brace, (>
L119:   <identifier, fptr1>
L119:   <brace, )>
L119:   <brace, )>
L119:   <relop, !=>
L119:   <identifier, EOF>
L119:   <brace, )>
L120:   <brace, {>
L120:   <identifier, num>
L120:   <arith-op, -->
L120:   <delimeter, ;>
L121:   <keyword, for>
L121:   <brace, (>
L121:   <identifier, weight>
L121:   <assign-op, =>
L121:   <integer, 128>
L121:   <delimeter, ;>
L121:   <identifier, weight>
L121:   <relop, >>
L121:   <integer, 0>
L121:   <delimeter, ;>
L121:   <identifier, weight>
L121:   <assign-op, =>
L121:   <identifier, weight>
L121:   <arith-op, />
L121:   <integer, 2>
L121:   <brace, )>
L122:   <brace, {>
L122:   <identifier, m>
L122:   <assign-op, =>
L122:   <identifier, c>
L122:   <arith-op, />
L122:   <identifier, weight>
L122:   <delimeter, ;>
L123:   <identifier, c>
L123:   <assign-op, =>
L123:   <identifier, c>
L123:   <arith-op, %>
L123:   <identifier, weight>
L123:   <delimeter, ;>
L124:   <keyword, if>
L124:   <brace, (>
L124:   <identifier, m>
L124:   <relop, ==>
L124:   <integer, 0>
L124:   <brace, )>
L125:   <identifier, p>
L125:   <assign-op, =>
L125:   <identifier, HT>
L125:   <brace, [>
L125:   <identifier, p>
L125:   <brace, ]>
L125:   <postfix operator, .>
L125:   <identifier, lchild>
```

```
L125:   <delimeter, ;>
L126:   <keyword, if>
L126:   <brace, (>
L126:   <identifier, m>
L126:   <relop, ==>
L126:   <integer, 1>
L126:   <brace, )>
L127:   <identifier, p>
L127:   <assign-op, =>
L127:   <identifier, HT>
L127:   <brace, [>
L127:   <identifier, p>
L127:   <brace, ]>
L127:   <postfix operator, .>
L127:   <identifier, rchild>
L127:   <delimeter, ;>
L128:   <keyword, if>
L128:   <brace, (>
L128:   <identifier, HT>
L128:   <brace, [>
L128:   <identifier, p>
L128:   <brace, ]>
L128:   <postfix operator, .>
L128:   <identifier, lchild>
L128:   <relop, ==>
L128:   <arith-op, ->
L128:   <integer, 1>
L128:   <logic-op, &&>
L128:   <identifier, HT>
L128:   <brace, [>
L128:   <identifier, p>
L128:   <brace, ]>
L128:   <postfix operator, .>
L128:   <identifier, rchild>
L128:   <relop, ==>
L128:   <arith-op, ->
L128:   <integer, 1>
L128:   <brace, )>
L129:   <brace, {>
L129:   <identifier, fputc>
L129:   <brace, (>
L129:   <identifier, p>
L129:   <delimeter, ,>
L129:   <identifier, fptr2>
L129:   <brace, )>
L129:   <delimeter, ;>
L130:   <identifier, p>
L130:   <assign-op, =>
L130:   <identifier, i>
L130:   <delimeter, ;>
L131:   <keyword, if>
L131:   <brace, (>
L131:   <identifier, num>
L131:   <relop, ==>
L131:   <arith-op, ->
L131:   <integer, 1>
L131:   <logic-op, &&>
L131:   <identifier, weight>
```

```
L131:   <relop, ==>
L131:   <identifier, pos>
L131:   <arith-op, *>
L131:   <integer, 2>
L131:   <brace, )>
L132:   <keyword, break>
L132:   <delimeter, ;>
L132:   <brace, }>
L133:   <brace, }>
L134:   <brace, }>
L135:   <identifier, printf>
L135:   <brace, (>
L135:   <string, "哈夫曼编码结束,␣输出到文件">
L135:   <brace, )>
L135:   <delimeter, ;>
L136:   <identifier, fclose>
L136:   <brace, (>
L136:   <identifier, fptr2>
L136:   <brace, )>
L136:   <delimeter, ;>
L137:   <keyword, return>
L137:   <delimeter, ;>
L138:   <brace, }>
L140:   <keyword, int>
L140:   <identifier, main>
L140:   <brace, (>
L140:   <keyword, void>
L140:   <brace, )>
L141:   <brace, {>
L141:   <keyword, int>
L141:   <identifier, n>
L141:   <delimeter, ,>
L141:   <identifier, i>
L141:   <delimeter, ,>
L141:   <identifier, j>
L141:   <delimeter, ,>
L141:   <identifier, k>
L141:   <delimeter, ,>
L141:   <identifier, weight>
L141:   <delimeter, ,>
L141:   <identifier, pos>
L141:   <delimeter, ,>
L141:   <identifier, num>
L141:   <assign-op, =>
L141:   <integer, 0>
L141:   <delimeter, ;>
L142:   <keyword, char>
L142:   <identifier, c>
L142:   <delimeter, ,>
L142:   <identifier, ch>
L142:   <delimeter, ;>
L143:   <keyword, char>
L143:   <arith-op, *>
L143:   <arith-op, *>
L143:   <identifier, HC>
L143:   <delimeter, ;>
L144:   <identifier, FILE>
L144:   <arith-op, *>
```

```
L144:   <identifier, fptr0>
L144:   <delimeter, ;>
L145:   <identifier, FILE>
L145:   <arith-op, *>
L145:   <identifier, fptr1>
L145:   <delimeter, ;>
L146:   <keyword, if>
L146:   <brace, (>
L146:   <brace, (>
L146:   <identifier, fptr0>
L146:   <assign-op, =>
L146:   <identifier, fopen>
L146:   <brace, (>
L146:   <string, "out0.txt","w+">
L146:   <brace, )>
L146:   <brace, )>
L146:   <relop, ==>
L146:   <identifier, NULL>
L146:   <brace, )>
L147:   <identifier, printf>
L147:   <brace, (>
L147:   <string, "can't␣open␣the␣object␣file\n">
L147:   <brace, )>
L147:   <delimeter, ;>
L148:   <keyword, if>
L148:   <brace, (>
L148:   <brace, (>
L148:   <identifier, fptr1>
L148:   <assign-op, =>
L148:   <identifier, fopen>
L148:   <brace, (>
L148:   <string, "out1.txt","wb+">
L148:   <brace, )>
L148:   <brace, )>
L148:   <relop, ==>
L148:   <identifier, NULL>
L148:   <brace, )>
L149:   <identifier, printf>
L149:   <brace, (>
L149:   <string, "can't␣open␣the␣object␣file\n">
L149:   <brace, )>
L149:   <delimeter, ;>
L150:   <identifier, printf>
L150:   <brace, (>
L150:   <string, "对象文件无法打开">
L150:   <brace, )>
L150:   <delimeter, ;>
L151:   <keyword, char>
L151:   <identifier, filename>
L151:   <brace, [>
L151:   <integer, 100>
L151:   <brace, ]>
L151:   <delimeter, ;>
L152:   <identifier, gets>
L152:   <brace, (>
L152:   <identifier, filename>
L152:   <brace, )>
L152:   <delimeter, ;>
```

```
L153:   <identifier, n>
L153:   <assign-op, =>
L153:   <identifier, CreateHuffmantree>
L153:   <brace, (>
L153:   <identifier, filename>
L153:   <brace, )>
L153:   <delimeter, ;>
L154:   <keyword, if>
L154:   <brace, (>
L154:   <identifier, n>
L154:   <relop, ==>
L154:   <integer, 0>
L154:   <brace, )>
L155:   <keyword, return>
L155:   <integer, 0>
L155:   <delimeter, ;>
L156:   <keyword, if>
L156:   <brace, (>
L156:   <identifier, n>
L156:   <relop, ==>
L156:   <integer, 1>
L156:   <brace, )>
L157:   <brace, {>
L157:   <identifier, rewind>
L157:   <brace, (>
L157:   <identifier, fptr>
L157:   <brace, )>
L157:   <delimeter, ;>
L158:   <keyword, while>
L158:   <brace, (>
L158:   <brace, (>
L158:   <identifier, c>
L158:   <assign-op, =>
L158:   <identifier, fgetc>
L158:   <brace, (>
L158:   <identifier, fptr>
L158:   <brace, )>
L158:   <brace, )>
L158:   <relop, !=>
L158:   <identifier, EOF>
L158:   <brace, )>
L159:   <brace, {>
L159:   <identifier, k>
L159:   <arith-op, ++>
L159:   <delimeter, ;>
L160:   <identifier, ch>
L160:   <assign-op, =>
L160:   <identifier, c>
L160:   <delimeter, ;>
L161:   <identifier, fputc>
L161:   <brace, (>
L161:   <char, '0'>
L161:   <delimeter, ,>
L161:   <identifier, fptr0>
L161:   <brace, )>
L161:   <delimeter, ;>
L162:   <keyword, if>
L162:   <brace, (>
```

```
L162:   <identifier, k>
L162:   <arith-op, %>
L162:   <integer, 8>
L162:   <relop, ==>
L162:   <integer, 0>
L162:   <brace, )>
L163:   <identifier, fputc>
L163:   <brace, (>
L163:   <brace, (>
L163:   <keyword, char>
L163:   <brace, )>
L163:   <integer, 0>
L163:   <delimeter, ,>
L163:   <identifier, fptr1>
L163:   <brace, )>
L163:   <delimeter, ;>
L163:   <brace, }>
L164:   <identifier, fputc>
L164:   <brace, (>
L164:   <brace, (>
L164:   <keyword, char>
L164:   <brace, )>
L164:   <integer, 0>
L164:   <delimeter, ,>
L164:   <identifier, fptr1>
L164:   <brace, )>
L164:   <delimeter, ;>
L164:   <brace, }>
L166:   <keyword, else>
L167:   <brace, {>
L167:   <identifier, HC>
L167:   <assign-op, =>
L167:   <identifier, CreateHuffmancode>
L167:   <brace, (>
L167:   <identifier, n>
L167:   <brace, )>
L167:   <delimeter, ;>
L169:   <keyword, while>
L169:   <brace, (>
L169:   <brace, (>
L169:   <identifier, i>
L169:   <assign-op, =>
L169:   <identifier, fgetc>
L169:   <brace, (>
L169:   <identifier, fptr>
L169:   <brace, )>
L169:   <brace, )>
L169:   <relop, !=>
L169:   <identifier, EOF>
L169:   <brace, )>
L170:   <identifier, fputs>
L170:   <brace, (>
L170:   <identifier, HC>
L170:   <brace, [>
L170:   <identifier, i>
L170:   <brace, ]>
L170:   <delimeter, ,>
L170:   <identifier, fptr0>
```

```
L170:   <brace, )>
L170:   <delimeter, ;>
L173:   <identifier, rewind>
L173:   <brace, (>
L173:   <identifier, fptr0>
L173:   <brace, )>
L173:   <delimeter, ;>
L174:   <identifier, weight>
L174:   <assign-op, =>
L174:   <integer, 128>
L174:   <delimeter, ;>
L175:   <identifier, j>
L175:   <assign-op, =>
L175:   <integer, 0>
L175:   <delimeter, ;>
L176:   <keyword, while>
L176:   <brace, (>
L176:   <brace, (>
L176:   <identifier, i>
L176:   <assign-op, =>
L176:   <identifier, fgetc>
L176:   <brace, (>
L176:   <identifier, fptr0>
L176:   <brace, )>
L176:   <brace, )>
L176:   <relop, !=>
L176:   <identifier, EOF>
L176:   <brace, )>
L177:   <brace, {>
L177:   <identifier, j>
L177:   <arith-op, +=>
L177:   <brace, (>
L177:   <identifier, i>
L177:   <arith-op, ->
L177:   <integer, 48>
L177:   <brace, )>
L177:   <arith-op, *>
L177:   <identifier, weight>
L177:   <delimeter, ;>
L178:   <identifier, weight>
L178:   <assign-op, =>
L178:   <identifier, weight>
L178:   <arith-op, />
L178:   <integer, 2>
L178:   <delimeter, ;>
L179:   <keyword, if>
L179:   <brace, (>
L179:   <identifier, weight>
L179:   <relop, ==>
L179:   <integer, 0>
L179:   <brace, )>
L180:   <brace, {>
L180:   <identifier, weight>
L180:   <assign-op, =>
L180:   <integer, 128>
L180:   <delimeter, ;>
L181:   <identifier, fputc>
L181:   <brace, (>
```

```
L181:   <brace, (>
L181:   <keyword, unsigned>
L181:   <keyword, char>
L181:   <brace, )>
L181:   <identifier, j>
L181:   <delimeter, ,>
L181:   <identifier, fptr1>
L181:   <brace, )>
L181:   <delimeter, ;>
L182:   <identifier, num>
L182:   <arith-op, ++>
L182:   <delimeter, ;>
L183:   <identifier, j>
L183:   <assign-op, =>
L183:   <integer, 0>
L183:   <delimeter, ;>
L183:   <brace, }>
L184:   <brace, }>
L185:   <keyword, if>
L185:   <brace, (>
L185:   <identifier, weight>
L185:   <relop, !=>
L185:   <integer, 0>
L185:   <brace, )>
L186:   <brace, {>
L186:   <identifier, fputc>
L186:   <brace, (>
L186:   <identifier, j>
L186:   <delimeter, ,>
L186:   <identifier, fptr1>
L186:   <brace, )>
L186:   <delimeter, ;>
L187:   <identifier, pos>
L187:   <assign-op, =>
L187:   <identifier, weight>
L187:   <delimeter, ;>
L187:   <brace, }>
L187:   <brace, }>
L188:   <identifier, printf>
L188:   <brace, (>
L188:   <string, "处理过程如下:\n">
L188:   <brace, )>
L188:   <delimeter, ;>
L189:   <identifier, c>
L189:   <assign-op, =>
L189:   <identifier, getchar>
L189:   <brace, (>
L189:   <brace, )>
L189:   <delimeter, ;>
L190:   <keyword, if>
L190:   <brace, (>
L190:   <identifier, c>
L190:   <relop, ==>
L190:   <char, 'y'>
L190:   <brace, )>
L191:   <keyword, if>
L191:   <brace, (>
L191:   <identifier, n>
```

```
L191:   <relop, >>
L191:   <integer, 1>
L191:   <brace, )>
L192:   <brace, {>
L193:   <identifier, Huffmandecoding>
L193:   <brace, (>
L193:   <identifier, fptr1>
L193:   <delimeter, ,>
L193:   <identifier, num>
L193:   <delimeter, ,>
L193:   <identifier, pos>
L193:   <delimeter, ,>
L193:   <identifier, filename>
L193:   <delimeter, ,>
L193:   <identifier, filetype>
L193:   <brace, (>
L193:   <identifier, filename>
L193:   <brace, )>
L193:   <brace, )>
L193:   <delimeter, ;>
L194:   <identifier, free>
L194:   <brace, (>
L194:   <identifier, HT>
L194:   <brace, )>
L194:   <delimeter, ;>
L195:   <keyword, for>
L195:   <brace, (>
L195:   <identifier, i>
L195:   <assign-op, =>
L195:   <integer, 0>
L195:   <delimeter, ;>
L195:   <identifier, i>
L195:   <relop, <>
L195:   <integer, 256>
L195:   <delimeter, ;>
L195:   <identifier, i>
L195:   <arith-op, ++>
L195:   <brace, )>
L196:   <identifier, free>
L196:   <brace, (>
L196:   <identifier, HC>
L196:   <brace, [>
L196:   <identifier, i>
L196:   <brace, ]>
L196:   <brace, )>
L196:   <delimeter, ;>
L197:   <identifier, free>
L197:   <brace, (>
L197:   <identifier, HC>
L197:   <brace, )>
L197:   <delimeter, ;>
L197:   <brace, }>
L198:   <keyword, else>
L199:   <brace, {>
L199:   <identifier, rewind>
L199:   <brace, (>
L199:   <identifier, fptr1>
L199:   <brace, )>
```

```
L199:   <delimeter, ;>
L200:   <identifier, FILE>
L200:   <arith-op, *>
L200:   <identifier, fptr2>
L200:   <delimeter, ;>
L201:   <keyword, char>
L201:   <identifier, objectfile>
L201:   <brace, [>
L201:   <integer, 10>
L201:   <brace, ]>
L201:   <assign-op, =>
L201:   <string, "out2">
L201:   <delimeter, ;>
L202:   <identifier, strcat>
L202:   <brace, (>
L202:   <identifier, objectfile>
L202:   <delimeter, ,>
L202:   <bitop, &>
L202:   <identifier, filename>
L202:   <brace, [>
L202:   <identifier, filetype>
L202:   <brace, (>
L202:   <identifier, filename>
L202:   <brace, )>
L202:   <brace, ]>
L202:   <brace, )>
L202:   <delimeter, ;>
L202:   <delimeter, ;>
L203:   <keyword, if>
L203:   <brace, (>
L203:   <brace, (>
L203:   <identifier, fptr2>
L203:   <assign-op, =>
L203:   <identifier, fopen>
L203:   <brace, (>
L203:   <identifier, objectfile>
L203:   <delimeter, ,>
L203:   <string, "wb">
L203:   <brace, )>
L203:   <brace, )>
L203:   <relop, ==>
L203:   <identifier, NULL>
L203:   <brace, )>
L204:   <identifier, printf>
L204:   <brace, (>
L204:   <string, "can't␣open␣the␣object␣file\n">
L204:   <brace, )>
L204:   <delimeter, ;>
L205:   <keyword, for>
L205:   <brace, (>
L205:   <identifier, i>
L205:   <assign-op, =>
L205:   <integer, 0>
L205:   <delimeter, ;>
L205:   <identifier, i>
L205:   <relop, <>
L205:   <identifier, k>
L205:   <delimeter, ;>
```

```
L205:    <identifier, i>
L205:    <arith-op, ++>
L205:    <brace, )>
L206:    <identifier, fputc>
L206:    <brace, (>
L206:    <identifier, ch>
L206:    <delimeter, ,>
L206:    <identifier, fptr2>
L206:    <brace, )>
L206:    <delimeter, ;>
L207:    <identifier, printf>
L207:    <brace, (>
L207:    <string, "处理结果写入到out2中">
L207:    <brace, )>
L207:    <delimeter, ;>
L208:    <identifier, fclose>
L208:    <brace, (>
L208:    <identifier, fptr2>
L208:    <brace, )>
L208:    <delimeter, ;>
L208:    <brace, }>
L211:    <identifier, fclose>
L211:    <brace, (>
L211:    <identifier, fptr>
L211:    <brace, )>
L211:    <delimeter, ;>
L212:    <identifier, fclose>
L212:    <brace, (>
L212:    <identifier, fptr0>
L212:    <brace, )>
L212:    <delimeter, ;>
L213:    <identifier, fclose>
L213:    <brace, (>
L213:    <identifier, fptr1>
L213:    <brace, )>
L213:    <delimeter, ;>
L214:    <keyword, return>
L214:    <integer, 0>
L214:    <delimeter, ;>
L215:    <brace, }>
```

## 7.4 分析和总结

通过 C++ 版和 FLEX 版分析结果的对照, 说明该 C 语言词法分析程序有以下功能:

1. 可以识别出用 C 语言编写的源程序中的每个单词符号, 运算符, 数字等, 并以 < 记号, 属性 > 的形式输出每个单词符号, 包括转义字符;
2. 可以识别并跳过源程序中的注释;
3. 可以统计源程序中的语句行数, 各类单词个数, 字符总数, 并输出统计结果;
4. 可以识别 C 语言程序源代码中存在的词法错误, 并报告错误出现的位置;

5. 对源程序中出现的错误进行适当的恢复, 让词法分析可以继续进行;
6. 对源程序进行一次扫描, 即可检查并报告源程序中存在的所有词法错误, 并输出源程序中所有的记号.

# 8 心得体会

通过这次词法分析程序的上手实践, 让我对编译原理课程的认识增加了除理论知识之外的内容, 亲自动手实现词法分析程序也让我对编译器进行词法分析的过程有了更加深入的理解.

词法分析程序与语法分析程序之间的关系可以有 3 种, 分别是词法分析程序作为独立的一遍, 词法分析程序作为语法分析程序的子程序, 和词法分析程序和语法分析程序作为协同程序. 在本次课程设计中, 将词法分析程序作为了独立的一遍, 可以将词法分析程序的输出放入到单独的中间文件, 让之后的语法分析程序读取中间文件即可获得词法分析结果, 有利于提柜编译程序的效率.

此外, 通过本次课程设计, 通过自己的动手亲身体验了将词法分析和语法分析等过程独立处理的好处: 如可以将各部分需要实现的功能进行良好封装和解耦合, 对外只暴露接口和提供服务, 各模块的具体实现对外部不可见, 简化了各部分实现的时候需要考虑的内容, 从而在实现识别并去除空格, 注释等功能的时候思路更加清晰, 还可以让程序可移植性, 可扩展性更强.

再次, 在本次课程设计中我还尝试了利用 FLEX 自动生成词法分析程序, 在 FLEX 自动生成版本和自己书写的版本的对比中, 体会到了 FLEX 功能的强大, 灵活和便利, 令我受益匪浅.

总体来说, 在本次课程设计过程中, 我对上学期所学形式语言和自动机知识, 以及本学期所学的词法分析内容都有了更加深刻的理解, 并掌握了运用方法; 此外, 编程能力, 程序设计能力等也有了不小的提升.

# 9 附录

代码仓库: Micuks/Lexical_Analysis