



哈爾濱工業大學(威海)

Harbin Institute of Technology at Weihai

编译原理实验报告

院 系： 计算机科学与技术学院
班 级： 2104102
姓 名： 蒲海博
学 号： 2021211041
指导教师： 闫健恩

哈尔滨工业大学（威海）

实验报告撰写规范

- 1) 正文使用五号字，单倍行距，中文使用宋体，英文使用 **times new roma** 字体；
- 2) 表格使用三线表，必须有表格标题；
- 3) 所有示意图必须使用 **visio**、**word** 等工具生成，不得手工画图或从其他资料中截图，并且每个图要有标题和编号。图中文字不得大于正文文字；
- 4) 注意行文的规范，段落空行等；
- 5) 报告采用双面打印，将 3 个实验一起装订，使用左侧装订方式；
- 6) 实验报告电子版请提供 **pdf** 格式。

《编译原理》实验成绩评定表

实验一 词法扫描器设计（共 20 分）			
评审项目	评审内容	成绩标准	成绩
实验内容完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	10 分	
系统设计	系统整体功能的分析设计与说明	5 分	
报告内容完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	5 分	
实验二 LR 语法分析器设计（共 40 分）			
评审项目	评审内容	成绩标准	成绩
实验内容完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	20 分	
系统设计	系统整体功能的分析设计与说明	10 分	
报告内容完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	10 分	
实验三 语义分析及中间代码生成（共 40 分）			
评审项目	评审内容	成绩标准	成绩
实验内容完成情况	实验内容是否全部完成； 程序功能是否完整； 实验结果是否准确； 调试分析是否完整等	20 分	
系统设计	系统整体功能的分析设计与说明	10 分	
报告内容完整性	文档格式规范性： 文字是否规范； 图表是否规范； 术语是否准确等	10 分	
总体评价		总成绩	

实验一 词法扫描器设计

一 实验目的

通过设计调试词法分析程序，实现从源程序中分出各种单词的方法；加深对课堂教学的理解；提高词法分析方法的实践能力。

二 实验内容

设计一个简单的类 C 语言的词法扫描器。

三 实验要求

（一）程序设计要求

- （1）根据附录给定的文法，从输入类 C 语言源程序中，识别出各个具有独立意义的单词，即关键字、标识符、常数、运算符、分隔符五大类；文法见最后附录。
- （2）提供源程序输入界面；
- （3）词法分析后可查看符号表和 TOKEN 串表；
- （4）保存符号表和 TOKEN 串表（如：文本文件）；
- （5）遇到错误时可显示提示信息，然后跳过错误部分继续进行分析。

```
int a;  
int b;  
int c;  
a=2;  
b=1;  
if (a>b)  
    c=a+b;  
else  
    c=a-b;
```

（二）实验报告撰写要求

- （1）系统功能（包括各个子功能模块的功能说明）；
- （2）开发平台（操作系统、设计语言）；
- （3）设计方案；
 - 1) 主程序流程图；
 - 2) 功能模块结构图；
 - 3) 主要子程序的流程图（若有必要）；
 - 4) 主要数据结构：符号表、TOKEN 串表等。
- （4）具体设计过程（包括主控程序、各个功能模块的具体实现）。

四 实验过程

（一）系统功能分析设计（包括各个子功能模块的功能说明）；

1) 读取源文件功能模块：

功能说明：

该模块负责读取指定的源代码文件，并将文件内容存储在一个字符串变量中供后续处理使用。

处理流程：

- ①使用 while 循环和 input_f.get(s);逐个字符地读取输入文件，并将其存储在 str 字符串中。
- ②当到达文件末尾时，退出循环。

2) 词法分析功能模块：

功能说明：

该模块实现了词法分析器，将源代码作为输入，对其进行遍历和分析，识别出各种不同类型的 Token（标记），并将它们添加到 Token 列表中。同时，对于识别出的标识符和关键字，会将其添加到符号表中。

处理流程：

- ①如果是数字，则将其视为一个整数，并存储在符号表和 token 数组中。
- ②如果是字母，则检查它是否是关键字或标识符，并相应地存储在符号表和 token 数组中。
- ③如果是操作符，则检查它是单字符操作符还是双字符操作符（如"=="），并存储在符号表和 token 数组中。
- ④如果是分隔符（如逗号、分号等），则存储在符号表和 token 数组中。
- ⑤如果是空格、换行或制表符，则跳过。
- ⑥对于其他任何字符，视为错误输入并打印错误消息。

3) 输出 Token 列表和符号表功能模块：

功能说明：

该模块负责将生成的 Token 列表和符号表打印到控制台，将符号表的内容写入 "signal_table.txt" 文件，将 token 数组的内容写入 "token.txt" 文件。

处理流程：

- ① 遍历 Token 列表，将每个 Token 的类型和值打印到控制台，并将其写入输出文件。
- ②打印和写入 Token 列表结束后，打印符号表的表头（name、type、kind、value）到控制台和输出文件。
- ③ 遍历符号表，将每个符号的各个字段值打印到控制台和输出文件。
- ④关闭输出文件。

以上是对每个子功能模块的详细的功能说明和处理流程描述。通过这些功能模块的组合，可以实现一个简单的词法分析器，能够识别源代码中的关键字、标识符、运算符和分隔符，并构建符号表、生成 Token 列表和符号表，完成对给定的源代码进行词法分析，并生成 Token 列表和符号表。

（二）开发平台（操作系统、设计语言）；

操作系统：

该代码在 Windows 11, version 22H2 中开发。

开发平台和设计语言：：

Visual Studio Code；该代码使用 C++编写。

（三）设计方案；

（1）主程序流程图；

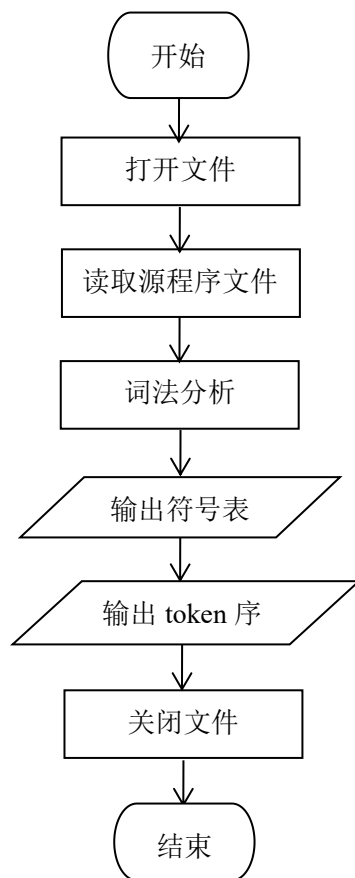


图 1-1 词法扫描器设计实验主程序流程图

(2) 功能模块结构图；

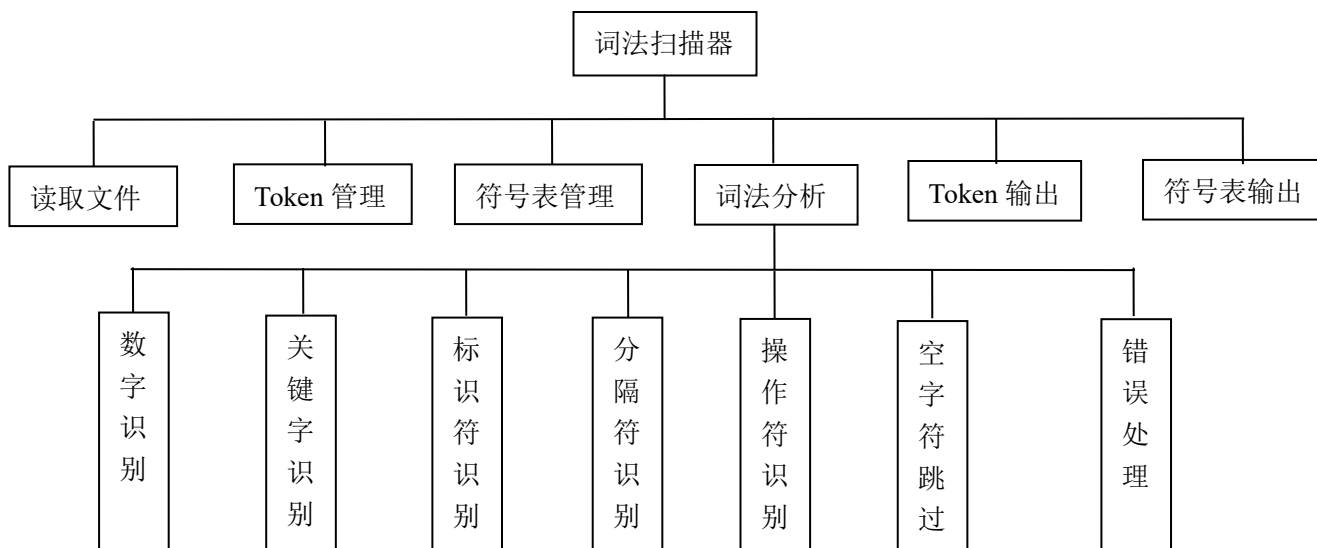


图 1-2 词法扫描器设计实验功能模块结构图

(三) 主要数据结构：符号表、TOKEN 串表等。

`string token[9999][2];`//token 串 种别，数值

`string signal[9999][4];`//符号表 名字，类型，种别，数值

```

int pos_token=0;
int pos_signal=0;
ifstream input_f;
ofstream signal_f,token_f;
token 和 signal//用于存储解析得到的 token 和符号表信息。
pos_token 和 pos_signal//分别表示当前 token 和符号的索引位置。
const unordered_map<string, string> keywords//无序哈希映射，用于存储关键字。
const unordered_map<char, string> operators//无序哈希映射，用于存储运算符。
const unordered_map<char, string> separators//无序哈希映射，用于存储分隔符。

```

这些数据结构在代码中用于存储和管理词法分析过程中生成的符号和 Token 信息。

（四）具体设计过程（包括主控程序、各个功能模块的具体实现）

主控程序的具体实现：

```

int main() {
    string token[MAX_TOKEN][2]; // token 串 种别, 数值
    string signal[MAX_SIGNAL][4]; // 符号表 名字, 类型, 种别, 数值
    int pos_token = 0;
    int pos_signal = 0;
    ifstream input_f(INPUT_FILE);
    ofstream signal_f(SIGNAL_TABLE_FILE);
    ofstream token_f(TOKEN_FILE);
    string str;
    getline(input_f, str);
    str = str.substr(0, str.length() - 1);
    int j = 0;
    int len = 0; // 当前单词的起始位置和长度
    while (j < str.length()) {
        len = 0;
        if (isDigit(str[j])) {
            while (isDigit(str[j + len])) len++;
            signal[pos_signal][0] = str.substr(j, len); // 名字
            signal[pos_signal][1] = "int"; // 类型
            signal[pos_signal][2] = "int10"; // 种属
            signal[pos_signal][3] = str.substr(j, len); // 数值
            pos_signal++;
            token[pos_token][0] = "int10";
            token[pos_token][1] = str.substr(j, len);
            pos_token++;
            j += len;
        } else if (isAlpha(str[j])) {
            while (isDigit(str[j + len]) || isAlpha(str[j + len])) len++;
            string keyword = isKeyword(str, j, len);
            if (keyword != "-1") { // 关键字
                signal[pos_signal][0] = str.substr(j, len); // 名字
                signal[pos_signal][1] = "char"; // 类型
            }
        }
    }
}

```

```

        signal[pos_signal][2] = "Keyword"; // 种属
        signal[pos_signal][3] = keyword; // 数值
        pos_signal++;
        token[pos_token][0] = "Keyword";
        token[pos_token][1] = keyword;
        pos_token++;
        j += len;
    } else { // 标识符
        signal[pos_signal][0] = str.substr(j, len); // 名字
        signal[pos_signal][1] = "char"; // 类型
        signal[pos_signal][2] = "id"; // 种属
        signal[pos_signal][3] = str.substr(j, len); // 数值
        pos_signal++;
        token[pos_token][0] = "id";
        token[pos_token][1] = str.substr(j, len);
        pos_token++;
        j += len;
    }
} else if (isOp(str[j], str[j + 1]) != "-1") {
    string ans = isOp(str[j], str[j + 1]);
    if (ans == "DOUBLE_EQ" || ans == "GET" || ans == "LET" || ans == "NEQ") {
        signal[pos_signal][0] = str.substr(j, 2); // 名字
        signal[pos_signal][1] = "char"; // 类型
        signal[pos_signal][2] = "OP"; // 种属
        signal[pos_signal][3] = ans; // 数值
        pos_signal++;
        token[pos_token][0] = "OP";
        token[pos_token][1] = ans;
        pos_token++;
        j += 2;
    } else {
        signal[pos_signal][0] = str[j]; // 名字
        signal[pos_signal][1] = "char"; // 类型
        signal[pos_signal][2] = "OP"; // 种属
        signal[pos_signal][3] = ans; // 数值
        pos_signal++;
        token[pos_token][0] = "OP";
        token[pos_token][1] = ans;
        pos_token++;
        j++;
    }
} else if (isSep(str[j]) != "-1") {
    string ans = isSep(str[j]);
    signal[pos_signal][0] = str[j]; // 名字

```



```

        signal[pos_signal][1] = "char"; // 类型
        signal[pos_signal][2] = "SEP"; // 种属
        signal[pos_signal][3] = ans; // 数值
        pos_signal++;
        token[pos_token][0] = "SEP";
        token[pos_token][1] = ans;
        pos_token++;
        j++;
    } else if (str[j] == ' ' || str[j] == '\n' || str[j] == '\t') { // 跳过空格、换行、tab
        j++;
    } else { // 错误输入
        cerr << "Error: " << str[j] << " in " << j << endl;
        j++;
    }
}

signal_f << "名字\t\t 类型\t\t 种属\t\t 数值" << endl;
for (int a = 0; a < pos_signal; ++a) {
    signal_f << signal[a][0] << "\t\t" << signal[a][1] << "\t\t" << signal[a][2] << "\t\t" << signal[a][3] << "\t\t" << endl;
}

for (int b = 0; b < pos_token; ++b) {
    token_f << token[b][0] << "\t\t" << token[b][1] << endl;
}

input_f.close();
signal_f.close();
token_f.close();
return 0;
}

```

词法分析程序的具体实现：

```

bool isAlpha(char t)判断字母 {
    return (t >= 'a' && t <= 'z') || (t >= 'A' && t <= 'Z');
}

```

```

bool isDigit(char t)判断数字 {
    return (t >= '0' && t <= '9');
}

```

```

string isKeyword 判断关键字(string str, int head, int len) {
    string sub = str.substr(head, len);
    if (sub == "if") return "if";
    else if (sub == "then") return "then";
    else if (sub == "else") return "else";
    else if (sub == "while") return "while";
    else if (sub == "do") return "do";
    else if (sub == "int") return "int";
}

```

```

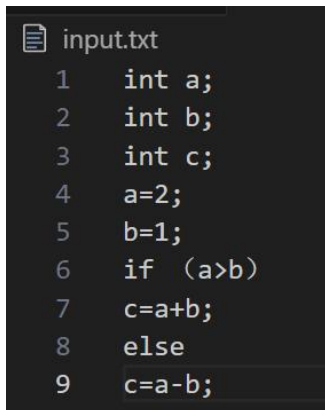
        return "-1";
    }

string isOp(char t1, char t2)判断操作符 {
    if (t1 == '=' && t2 == '=') return "DOUBLE_EQ";
    else if (t1 == '>' && t2 == '=') return "GET";
    else if (t1 == '<' && t2 == '=') return "LET";
    else if (t1 == '!' && t2 == '=') return "NEQ";
    else if (t1 == '+') return "ADD";
    else if (t1 == '-') return "SUB";
    else if (t1 == '*') return "MUL";
    else if (t1 == '/') return "DIV";
    else if (t1 == '=') return "EQ";
    else if (t1 == '>') return "GT";
    else if (t1 == '<') return "LT";
    return "-1";
}

string isSep(char t) {
    if (t == "\") return "QUO";
    else if (t == '(') return "LP";
    else if (t == ')') return "RP";
    else if (t == ';') return "SEMI";
    return "-1";
}

```

（五）程序运行结果



```

input.txt
1   int a;
2   int b;
3   int c;
4   a=2;
5   b=1;
6   if (a>b)
7       c=a+b;
8   else
9       c=a-b;

```

图 1-3 测试程序图

```

1.cpp token.txt X
> Users > pu > Desktop > 实验1 > token.txt
1 Keyword INT
2 id a
3 SEP SEMI
4 Keyword INT
5 id b
6 SEP SEMI
7 Keyword INT
8 id c
9 SEP SEMI
10 id a
11 OP EQ
12 id 2
13 SEP SEMI
14 id b
15 OP EQ
16 id 1
17 SEP SEMI
18 Keyword IF
19 id a
20 OP GT
21 id b
22 id c
23 OP EQ
24 id a
25 OP ADD
26 id b
27 SEP SEMI
28 Keyword ELSE
29 id c
30 OP EQ
31 id a
32 OP SUB
33 id b
34 SEP SEMI

```

图 1-4 token 输出表

```

signal_table.txt
1 a INT var
2 b INT var
3 c INT var
4 2 INT CONST 2
5 1 INT CONST 1

```

图 1-5 符号输出表

五 实验总结

实验完成了词法分析器，将输入的源代码按照词法规则进行解析，生成对应的 token 和符号表。根据预定义的关键字、操作符和分隔符进行识别和分类。输入源代码经过词法分析器处理后，生成了 token 二元组。结果输出到了相应的文件中（token.txt 和 signal_table.txt）。经过检查输出文件，token 和符号表符合预期结果。对于关键字、操作符和分隔符的识别分类都能够正常进行。

但在处理大规模源代码时，需要考虑程序是否能够有效地进行词法分析，是否存在性能瓶颈。是否需要程序进行优化以提高效率。

可以改进的方面：是否可以添加更多的词法规则，以适应更多语言的词法分析需求；是否可以优化现有代码，提高代码的可读性和可维护性；是否可以添加错误处理机制，以应对输入源代码中可能存在的错误。

通过实验，我验证了词法分析器的基本功能，并对其有更深入的了解。在实际应用中，词法分析器对于编译器等程序的构建具有重要意义，能够帮助程序理解源代码的结构和含义，为后续的语法分析和语义分析奠定基础。

实验二 LR语法分析器设计

一 实验目的

通过设计调试 LR 语法分析程序，实现根据词法分析的输入 TOKEN 字，进行文法的语法分析；加深对课堂教学的理解；提高语法分析方法的实践能力。

二 实验内容

使用附录中的文法，可以对类似下面的程序语句进行语法分析：

```
int a;  
int b;  
int c;  
a=2;  
b=1;  
if (a>b)  
    c=a+b;  
else  
    c=a-b;
```

三 实验要求

（一） 程序设计要求

- （1）给出主要数据结构：分析栈、符号表；
- （2）将词法扫描器作为一个子程序，每次调用返回一个 TOKEN；
- （3）程序界面：表达式输入、语法分析的表示结果（文件或者图形方式）；

（二）实验报告撰写要求

- （1）系统功能分析与设计（包括各个子功能模块的功能说明）；
- （2）开发平台（操作系统、设计语言）；
- （3）设计方案:包括功能模块结构图、主要函数的流程图；
- （4）主要数据结构：分析栈、分析表、符号表；
- （5）具体设计实现过程（包括主控程序、各个功能模块的具体实现）。

四 实验过程

（一） 系统功能分析设计（包括各个子功能模块的功能说明）；

1) 词法分析模块：

功能说明：

该模块实现了词法分析的功能，将输入的源代码转换为一个个的标记（token），并将其保存到文件中。

处理流程：

①通过遍历源代码字符流，根据预定义的规则判断字符的类型，如字母、数字、运算符、分隔符等，并将其组合成一个个的标记（token）。

②通过判断字符序列是否为预定义的关键字（如"if"、"else"等）或由字母和数字组成的标识符，来区分不同类型的标记。

③通过判断字符是否为运算符（如"+"、"-"等）或分隔符（如"("、")"等），来识别相应的标记。

2) LR 项集族构造模块：

功能说明：

实现了 LR(1)项集族的构造

处理流程：

①根据 LR(1)项的定义，对给定的项进行闭包操作，即将每个项中的可扩展非终结符扩展为可能的下一个状态。

②根据当前项集的状态和输入字符，确定下一个状态集合。

3) First 集生成模块：

功能说明：

该模块实现了 First 集的生成，用于语法分析过程中的预测分析表的构建。

处理流程：

① 通过深度优先搜索（DFS）算法遍历产生式，根据产生式右部的字符，判断其是否为终结符。

②若为终结符则加入 First 集，若为非终结符则继续向下递归计算。

4) Follow 集生成模块：

功能说明：

实现了 Follow 集的生成，也是预测分析表构建的一部分

处理流程：

①根据文法的定义，对每个非终结符进行 Follow 集的计算，根据产生式右部的情况，判断是否需要考虑 Follow 集的加入。

5) LR 分析表构建模块：

功能说明：

根据已经构造好的 LR(1)项集族、First 集和 Follow 集，按照 LR 分析表的格式，填充 ACTION 和 GOTO 表的内容

6) 语法分析模块：

功能说明：

根据填充好的 LR 分析表，对输入的 token 序列进行分析，按照 LR 分析算法的规则进行移入、归约和接受的操作，判断输入的源代码是否符合给定的语法规则。

以上是对每个子功能模块的详细的功能说明和处理流程描述。通过这些功能模块的组合，可以实现一个简单的语法分析器。

（二）开发平台（操作系统、设计语言）；

操作系统：

该代码在 Windows 11, version 22H2 中开发。

开发平台和设计语言：：

Visual Studio Code；该代码使用 C++编写。

（三）设计方案；

(1) 主程序流程图;

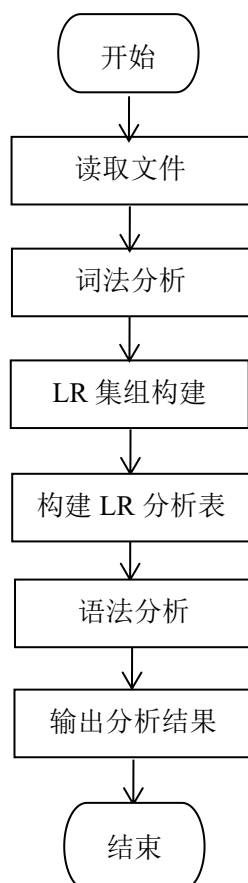


图 2-1 语法分析器设计实验主程序流程图

(2) 功能模块结构图;

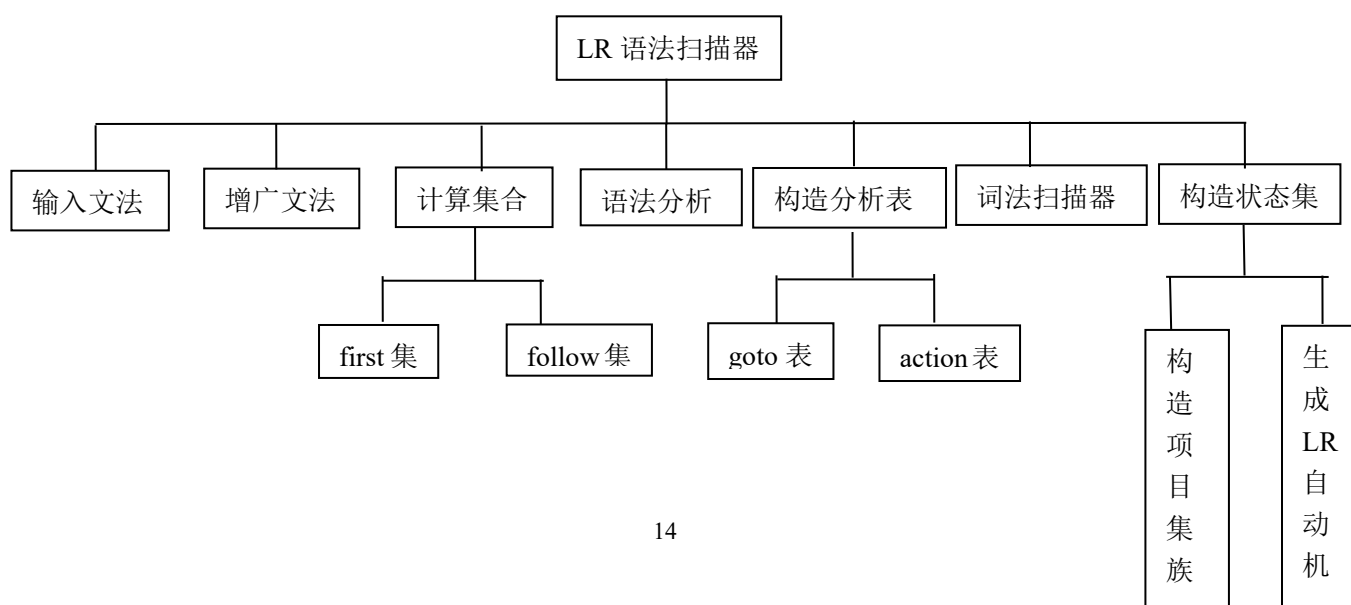


图 2-2 语法分析器设计实验功能模块结构图

(3) 主要数据结构:

```
vector<string> prod;//产生式
vector< vector<string> > ItemSet;//项集
char ACTION[999][30]={0}; action 表
int GOTO[999][30]={0};//goto 表
string map_char="BPDSLCEFTd;@i=f()ew><#+ -* /n$";//字符到数字的映射
vector<string> first;//first 集
vector<string> follow;//follow 集
int gotfirst[30]={0};//已生成 first 集
```

(四) 具体设计过程 (包括主控程序、各个功能模块的具体实现)

主控程序的具体实现:

```
int main(){

    read_gramma();//读入产生式
    make_ItemSet();//构造项集
    make_first();//求 first 集
    make_follow();//求 follow 集
    read_code();//生成 token
    make_analyze_table();//建立分析表
    analyze();//分析器
    return 0;
}
```

集组构造程序的具体实现:

```
void make_ItemSet(){
    ofstream itemset_f("项集.txt",ios::out);
    vector<string> status_now;
    status_now.push_back("B|.P");//初始项
    closuer(status_now);
    ItemSet.push_back(status_now);
    for(int i=0;i<ItemSet.size();++i){
        itemset_f<<"-----status "<<i<<"-----\n";
        for(int j=0;j<ItemSet[i].size();++j){
            itemset_f<<ItemSet[i][j]<<endl;
        }
        itemset_f<<"状态转移: \n";
        sta_out(ItemSet[i],i,itemset_f);
    }
    itemset_f.close();
}
```



```

        return ;
    }

void closuer(vector<string> &status_now){
    for(int i=0;i<status_now.size();++i){
        char next_char=find_next_char(status_now[i]); //.后的字符
        if(next_char!=-1&&next_char!=0){
            for(int k=0;k<prod.size();++k){
                if(prod[k][0]==next_char){
                    string t="";
                    t+=prod[k].substr(0,3);
                    t+=".";
                    t+=prod[k].substr(3);
                    add2vec(status_now,t);
                }
            }
        }
    }
    return ;
}

```

First 集生成程序的具体实现：

```

void make_first(){
    for(int i=0;i<map_char.size();++i){
        string t="";
        t+=map_char[i];
        if(map_char[i]<'A' || map_char[i]>'Z'){//终结符
            t+=map_char[i];
            gotfirst[i]=1;
        }
        first.push_back(t);
    }
    for(int i=0;i<map_char.size();++i){
        if(gotfirst[i]==1) continue;
        getfirst(i);//dfs 产生 first 集
    }
    ofstream first_f("first.txt",ios::out);
    for(int i=0;i<first.size();++i){
        first_f<<first[i]<<endl;
    }
    first_f.close();
    return ;
}

```

Follow 集生成程序的具体实现：

```
void make_follow(){
    for(int i=0;i<map_char.size();++i){
        if(map_char[i]<'A' || map_char[i]>'Z') continue;
        string t="";
        t+=map_char[i];
        if(i==0) t+="$";
        follow.push_back(t);
    }
    for(int i=0;i<map_char.size();++i){
        if(map_char[i]<'A' || map_char[i]>'Z') continue;
        if(gotfollow[i]==0) getfollow(i);
    }
    ofstream follow_f("follow.txt",ios::out);
    for(int i=0;i<follow.size();++i){
        follow_f<<follow[i]<<endl;
    }
    follow_f.close();
    return ;
}
```

LR 分析表构建程序的具体实现：

```
void make_analyze_table(){
    for(int i=0;i<ItemSet.size();++i){
        for(int j=0;j<ItemSet[i].size();++j){
            int pos=ItemSet[i][j].find('.');
            if(find_next_char(ItemSet[i][j])==-1 || find_next_char(ItemSet[i][j])==0){//在最右边
                if(ItemSet[i][j][0]=='B')//接受
                    ACTION[i][char2int('$')]='a';
                else{//归约
                    int loca=locat_follow(ItemSet[i][j][0]);//找到 follow 集
                    for(int k=1;k<follow[loca].size();++k){
                        ACTION[i][char2int(follow[loca][k])]='r';//reduce 归约
                        GOTO[i][char2int(follow[loca][k])]=prod_num(ItemSet[i][j].substr(0,ItemSet[i][j].size()-1));
                    }
                }
            }
            else if(find_next_char(ItemSet[i][j])!=0){
                if(ItemSet[i][j][pos+1]>='A' && ItemSet[i][j][pos+1]<='Z'){//非终结符
                    ACTION[i][char2int(ItemSet[i][j][pos+1])]='g';//goto
                    GOTO[i][char2int(ItemSet[i][j][pos+1])]=go[i][char2int(ItemSet[i][j][pos+1])];
                }
                else{//终结符
                    ACTION[i][char2int(ItemSet[i][j][pos+1])]='m';//move in
                    GOTO[i][char2int(ItemSet[i][j][pos+1])]=go[i][char2int(ItemSet[i][j][pos+1])];
                }
            }
        }
    }
}
```

```

        }
    }
}

ofstream ACTION_f("ACTION.txt",ios::out);
ofstream GOTO_f("GOTO.txt",ios::out);
ACTION_f<<map_char<<endl;
for(int i=0;i<ItemSet.size();++i){
    for(int j=0;j<30;++j){
        ACTION_f<<ACTION[i][j];
        GOTO_f<<GOTO[i][j];
    }
    ACTION_f<<endl;
    GOTO_f<<endl;
}
ACTION_f.close();
GOTO_f.close();
return ;
}

```

语法分析程序的具体实现：

```

void analyze(){
    ofstream analyze_f("analyze.txt",ios::out);
    stack<int> status_stack;//状态栈
    stack<char> chara_stack;//符号栈
    while(!status_stack.empty()) status_stack.pop();
    while(!chara_stack.empty()) chara_stack.pop();
    token+="$";
    status_stack.push(0);
    int pos=0;//输入串中的位置
    while(true){
        int status=status_stack.top();
        if(ACTION[status][ char2int(token[pos]) ]=='m'){//移入
            cout<<"移入：  "<<token[pos]<<"  位置：  "<<pos<<endl;
            analyze_f<<"移入：  "<<token[pos]<<"  位置：  "<<pos<<endl;
            chara_stack.push(token[pos]);
            status_stack.push(GOTO[status][ char2int(token[pos]) ]);
            show_stacks(analyze_f,status_stack,chara_stack);
            pos++;
        }
        else if(ACTION[status][ char2int(token[pos]) ]=='r'){//归约
            string prod_used=prod[ GOTO[status][ char2int(token[pos]) ] ];
            for(int i=0;i<prod_used.size()-3;++i){
                status_stack.pop();
                chara_stack.pop();
            }

```

```

    }
    chara_stack.push(prod_used[0]);
    status_stack.push( GOTO[status_stack.top()][ char2int(prod_used[0]) ] );
    cout<<"归约:  "<<prod_used<<endl;
    analyze_f<<"归约:  "<<prod_used<<endl;
    show_stacks(analyze_f,status_stack,chara_stack);
}
else if(ACTION[status][ char2int(token[pos]) ]=='a'){//接受
    cout<<"分析完成, 可以接受"<<endl;
    analyze_f<<"分析完成, 可以接受"<<endl;
    show_stacks(analyze_f,status_stack,chara_stack);
    break;
}
else {
    cout<<"a error: "<<token[pos]<<endl;
    analyze_f<<"a error: "<<token[pos]<<endl;
    show_stacks(analyze_f,status_stack,chara_stack);
    pos++;
    break;
}
}
analyze_f.close();
return ;
}

```

（五）程序运行结果

The screenshot shows a terminal window with the following output:

```

状态栈: 14    符号栈: ;
状态栈: 9     符号栈: d
状态栈: 3     符号栈: L
归约:  D->Ld;D
状态栈: 24    符号栈: D
状态栈: 14    符号栈: ;
状态栈: 9     符号栈: d
状态栈: 3     符号栈: L
归约:  D->Ld;D
状态栈: 2     符号栈: D
移入:  d  位置: 9
状态栈: 6     符号栈: d
状态栈: 2     符号栈: D
移入:  =  位置: 10
状态栈: 11    符号栈: =

```

图 2-3 语法分析器程序部分运行结果图

五 实验总结

通过这次实验，我更加熟悉了语法分析的过程，并结合课上所学的理论知识，对语法分析有了更深入的理解。

首先，根据输入的文法，我添加了状态和产生式，构造出增广文法。然后，利用得到的文法计算了 First 集和 Follow 集合。这里首先计算 First 集合是为了方便后续计算 Follow 集合。而计算 Follow 集合则是为了之后构造分析表。LR 分析法考虑了部分上下文，因此能够消除一部分的移入归约冲突，相比之下，LR(0)分析法则直接进行归约操作而不考虑上下文。接着，我开始构造项目集族。利用这些状态构成的 LR 自动机以及之前计算的 Follow 集，我构造了 Action 表和 Goto 表。有了 Action 表和 Goto 表，就可以进行语法分析了。在实验过程中，我遇到了一些问题，如构建 LR(1)项集族时的状态转移和 LR 分析表的构建等。通过查阅资料、思考和尝试，我逐步解决了这些问题，提升了自己的问题解决能力和编程技巧。通过这次实验，我对语法分析的整个过程有了更深入的理解。

实验三 语义分析及中间代码生成

一 实验目的

通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。

二 实验内容

实现简单的高级语言源程序的语义处理过程。

三 实验要求

（一）程序设计要求

- （1）目标机：8086 及其兼容处理器
- （2）中间代码：三地址码
- （3）主要数据结构：三地址码表、符号表
- （4）语义分析内容要求：
 - 1) 变量说明语句
 - 2) 赋值语句
 - 3) 控制语句(任选一种)
- （5）其它要求：
 - 1) 将词法分析（扫描器）作为子程序，供语法、语义程序调用；
 - 2) 使用语法制导的语义翻译方法；
 - 3) 编程语言自定；
 - 4) 提供源程序输入界面；
 - 5) 目标代码生成暂不做；
 - 6) 编译后，可查看 TOKEN 串、符号表、三地址码表；
 - 7) 主要数据结构：产生式表、符号表、三地址码表。

所用文法使用实验 2 中的文法。

附录：语义分析源程序范例

```
int a;
int b;
int c;
a=2;
b=1;
if (a>b)
    c=a+b;
else
    c=a-b;
```

（二）实验报告撰写要求

- (1) 系统功能分析与设计（包括各个子功能模块的功能说明）；
- (2) 开发平台（开发软硬件环境）；
- (3) 语义翻译中使用的数据结构；
- (4) 程序具体设计实现过程（包括主要功能模块的具体实现）。

四 实验过程

（一）系统功能分析设计（包括各个子功能模块的功能说明）；

1) 语句分析功能模块：

功能说明：

根据语法规则进行不同的处理，调用其他函数来处理不同类型的语句。

可以处理一组连续的语句。它通过循环调用函数来处理每个语句，直到遇到语句序列的结束。

2) 语法分析功能模块：

功能说明：

接受一段代码作为输入，并调用其他函数来进行词法分析、变量声明分析、语句处理等。是整个语法分析过程的控制中心，将代码分解为语法单元，并进行相应的处理和解析。

3) 词法分析功能模块：

功能说明：

按照一定的规则对代码进行扫描和解析。在扫描的过程中，它识别出不同的词法单元（例如关键字、标识符、常量、运算符等），并将它们保存到 token 表中，以备后续的语法分析使用。

（二）开发平台（操作系统、设计语言）；

操作系统：

该代码在 Windows 11, version 22H2 中开发。

开发平台和设计语言：：

Visual Studio Code；该代码使用 C++编写。

（三）主要数据结构：符号表、TOKEN 串表等。

```
char *Keyword[]=
{
    //关键字
    "const","int","char","void",
    "main","if","else","do","while",
    "scanf","printf","return"
};

char *oprator[]=
{
    //操作码
    ""," ","JME","JMC","JMP","CAS",
    "RED","EXF","WRI"," ",":","+","-","*",
    ":",*","./","MUS","j>","j<","j==",
    "j!=", "j>=", "j<=", "","CAL","BEGIN",
    "END",":="
};

union Cha{
    int num;
    char cischar;
```

```

};
struct Character
{//符号表
    char *name ;
    int type,lev,adr,kind;
    Cha uchar;
};
struct opr
{//中间代码，表示符号表中的条目的结构体。它包含了符号的名称（name）、类型（type）、
层级（lev）、地址（adr）、种类（kind）等信息。
    int op,op1,op2,result;
};
struct aopr
{
    int anum,ad1;
};
FILE *Threecode;//三地址代码
FILE *KeyWord;//关键字文件
FILE *Oprate;//操作符文件
FILE *Token;//token 文件

```

（四）具体设计过程（包括主控程序、各个功能模块的具体实现）

主控程序的具体实现：

```

int main(int argc,char *argv[])
{
    int c=0;
    init();
    FIN=fopen("code.txt","r");

    if(FIN==NULL)
    {
        printf("file not found!\n");
        exit(1);
    }

    printf("\n 源程序:\n");
    FOUT=stdout;

    for(int j=0;j<MAXLEV;j++)//DEFINE:MAXLEV 符号表的最大容量 50
        numoflev[j]=1;//各层的表中变量数

    if(program())
    {
        int i=errornum;

```



```

        for(errornum=0;errornum<i;errornum++)
        {
            error_msg(error[errornum]);
        }
        return 0;
    }
    printf("\n\n 三地址代码:\n");
    int i;
    Threecode=fopen("3_code.txt","wt+");
    for(i=0;i<oprnum;i++)
    {
        fprintf(FOUT,"%d\t%d\t=\t%d\t\t%s\t%d\n",i,opra[i].result,
            opra[i].op1,oprate[opra[i].op],opra[i].op2);
        fprintf(Threecode,"%d\t%d\t=\t%d\t\t%s\t%d\n",i,opra[i].result,
            opra[i].op1,oprate[opra[i].op],opra[i].op2);
    }
    fclose(Threecode);
    printf("\n\n 三地址代码输出完成!\n\n");
    getch();
}

```

语法分析程序的具体实现：

```

int analyzer()
{
    int i;
    char c;
    symbol_Id = getNextToken();

    if (symbol_Id == 11)//数字 11--int,
    {
        i = symbol_Id;
        symbol_Id = getNextToken();
        if (symbol_Id == 1)//标识符--1
        {
            strcpy_s(ids[symNum], token_id);//NumOfC 符号表内元素的个数 //name 存标识符
            symbol_Id = getNextToken();

            while (symbol_Id == '(')// (
            {
                symbol_Id = getNextToken();
                if (symbol_Id == 1)// 1--标识符
                {
                    strcpy_s(ids[symNum], token_id);
                    if ((symbol_Id = getNextToken()) == '(')/'(' (

```

```

                                continue;
                        }
                }

        }

    }

    if (symbol_Id == 13)//13--void
    {
        symbol_Id = getNextToken();
        if (symbol_Id == 14)//14--main
        {
            if (analyzeMain())//主函数分析
            {
                return 1;
            }
            else
                return 0;
        }
    }

    return 0;
}

```

```

int statements()
{
    int a = 0;
    while (symbol_Id == 1 || symbol_Id == 15)//1 标识符 15if
    {
        if ((a = statement()) == 1)
            return 1;
    }
    if (symbol_Id == ';' )// ;
        symbol_Id = getNextToken();
    return 0;
}

```

语句分析程序的具体实现：

```

int statement()
{
    int a;
    switch(sym)
    {
        case 15://15 if
            return(ifstatement());
        case 18://18 while
            return(whilestatement());
    }
}

```

```

case 19://19 scanf
    return(scanfstatement());
case 20://20 printf
    return(printfstatement());
case 1://1 标识符
    if((a=find())==0)
        return 1;
    sym=getsym();
    if(Char[a].kind==0||Char[a].kind==3)//0 为参数 3 为变量
    {
        if(sym==51)//51 为 =
            return(assignstatement(a));//赋值语句分析
        else
        {
            errorline[errornum]=linenum;
            error[errornum++]=9;
            return 1;
        }
    }
    else if(Char[a].kind==1||Char[a].kind==2)//1 有返回值 2 无有返回值
    {
        if(sym==37)//37 (
        {
            if(call(a))//函数调用
                return 1;
            if(sym==38)//38 ;
                sym=getsym();
            else
            {
                errorline[errornum]=linenum;
                error[errornum++]=22;
                return 1;
            }
            return 0;
        }
        else
        {
            errorline[errornum]=linenum;
            error[errornum++]=17;
            return 1;
        }
    }
    else
    {

```

```

        errorline[errornum]=linenum;
        error[errornum++]=10;
        return 1;
    }
case 40://40 {
    sym=getsym();
    if((a=statementArray())==1)
        return 1;
    if(sym!=35)//35  }
    {
        errorline[errornum]=linenum;
        error[errornum++]=12;
        return 1;
    }
    sym=getsym();
    return a;
case 38://38  ;
    sym=getsym();
    return 1;
default:
    return 1;
}
return 1;
}

```

（五）程序运行结果

```

0:  a   =   2
1:  b   =   1
2:  if  a>b goto    4
3:  goto 7
4:  t0  =  a+b
5:  c   =  t0
6:  goto 9
7:  t1  =  a-b
8:  c   =  t1

```

图 3-1 输出结果图

五 实验总结

这次实验展示了编译器的语法制导翻译的基本流程，结合了前两个实验的成果，利用了词法分析和语法分析得到的 token 序列、符号表和语法分析结果，进一步实现了语义分析和中间代码生成。

通过这个实验，我们深入理解了语法制导翻译的原理，并学会了如何将语法分析得到的语法结构转换为中间代码。词法分析、语法分析、语义处理和中间代码生成是编译器重要的组成部分，对于理解和构建编译器、学习编译原理都至关重要。与前两次实验相比，这次是最具挑战性的，因为需要处理复杂的语义分析并熟练操作数据结构来生成中间代码。在解决问题的过程中，我查阅了大量资料，收获颇丰。通过这次实验，我学会了如何设计、编写和调试语义分析程序，加深了对语法制导翻译原理的理解，掌握了将语法分析结果转换为中间代码的方法，对语义分析和中间代码生成有了更深入的理解。

通过对实验代码的理解和实践，我大大加深了对编译器工作原理的理解。