**编译原理实验一**

**201801130227 孙浩然**

1. **实验目的**

学习和掌握词法分析程序手工构造状态图及其代码实现方法。

1. **实验任务**
2. 阅读已有编译器的经典词法分析源程序；
3. 用C或C++语言编写一门语言的词法分析器。
4. **实验内容**
5. 阅读已有编译器的经典词法分析源程序。

选择一个编译器，如：TINY，其它编译器也可（需自备源代码）。阅读词法分析源程序，理解词法分析程序的手工构造方法——状态图代码化。尤其要求对相关函数与重要变量的作用与功能进行稍微详细的描述。若能加上学习心得则更好。TINY语言请参考《编译原理及实践》第2.5节（见压缩包里附带的文档）。

1. 确定今后其他实验中要设计编译器的语言，如TINY语言，又如更复杂的C－语言（其定义在《编译原理及实践》附录A中）。也可选择其它语言，不过要有该语言的详细定义（可仿照C－语言）。一旦选定，不能更改，因为要在以后继续实现编译器的其它部分。鼓励自己定义一门语言。
2. 根据该语言的关键词和识别的词法单元以及注释等，确定关键字表，画出所有词法单元和注释对应的DFA图。
3. 仿照前面学习的词法分析器，编写选定语言的词法分析器。
4. 准备2~3个测试用例，要求包含正例和反例，测试编译结果。

**实验步骤：**

1. 阅读TINY编译器的词法分析源程序
   1. getToken函数：

主要目的是进行DFA状态的转移，以及获取接受的字符串。首先定义了TokenType类型(globals.h中)，包含了两种特殊结束情况：ERROR和ENDFILE，之后定义了TINY语言的记号：8个保留字，10个特殊符号对应的String，以及两种其他情况：数字和标识符。在定义了currentToken表示这次接受的Token类型和当前状态state之后，开始进行DFA状态的转移，通过switch嵌套实现，具体过程就和画出的DFA图相同。其中要注意的细节是：当不匹配时要进行回退，状态设置为ERROR并将save设置为false，在之后不对tokenString进行修改。

* 1. getNextChar函数：

读入输入文件的一行存放在lineBuf内，设置一个指针指向当前读入的元素，并且每次读入新的一行都使行号加一，便于在输出时显示。如果指针指向了当前文本的末尾，则读入新的一行再返回当前字符，否则直接返回当前指向的字符即可，并且要使指针后移一位。

* 1. ungetNextChar函数：

当没有读入EOF的时候，便将上述的指针前移一位，实现回溯的操作。

1. 确定今后其他实验中要设计编译器的语言

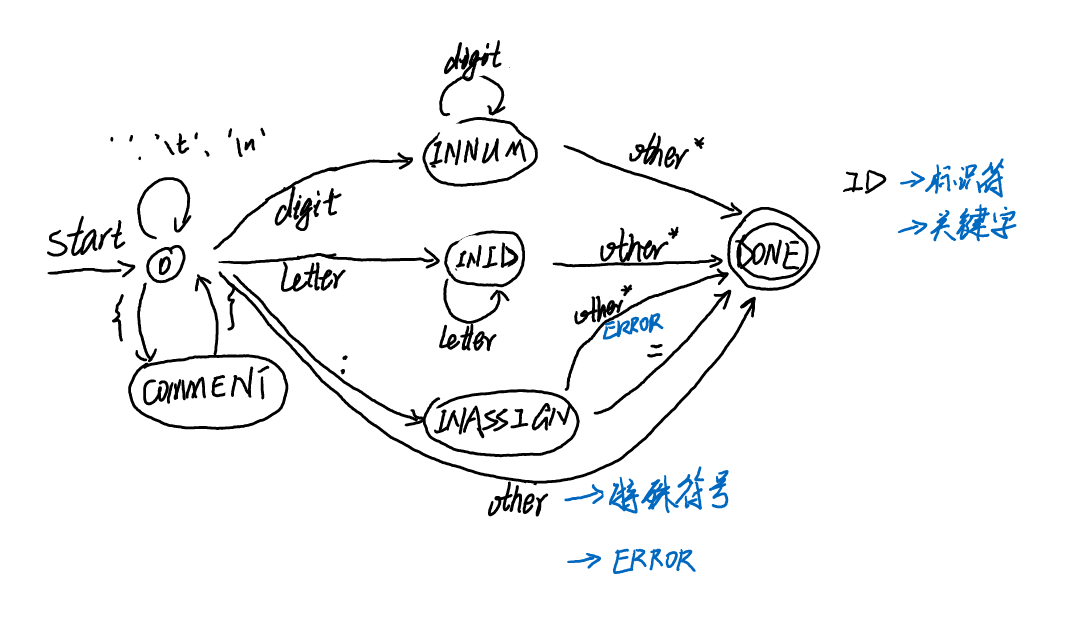
选择TINY语言

1. 确定关键字表，画出所有词法单元和注释对应的DFA图

关键字表：

|  |  |  |
| --- | --- | --- |
| 保留字 | 特殊符号 | 其他 |
| if | + | 数字NUM |
| then | - | 标识符ID |
| else | \* |  |
| end | / |  |
| repeat | = |  |
| until | < |  |
| read | > |  |
| write | ( |  |
|  | ) |  |
|  | ; |  |
|  | := |  |

DFA图：



1. 编译器的编写：

**变量声明：**

typedef enum {

    ENDFILE, ERROR,

    IF, THEN, ELSE, END, REPEAT, UNTIL, READ, WRITE,

    PLUS, MINUS, TIMES, DIVIDE, EQULE, LE, LB, RB, SEMI, ASSIGN,

    NUM, ID

}TokenType;

typedef enum {

    START, INNUM, INID, INASSIGN, DONE, INCOMMENT

}StateType;

struct {

    char\* str;

    TokenType token;

}reserved[8] =

{{"if",IF},{"then",THEN},{"else",ELSE},{"end",END},

{"repeat",REPEAT},{"until",UNTIL},{"read",READ},{"write",WRITE}};

int buf\_size = 0;

char buffer[MAX\_BUF\_SIZE];

int name\_pos = 0;

char currString[MAX\_STR\_SIZE];

FILE \*fp;

TokenType声明了以上关键字表中用到的变量别名

StateType声明了用到的6个状态

reserved数组用于检测是否为保留字，因为保留字和ID最终都会先被识别为ID，所以需要判断是不是保留字

buffer数组存放因为回退而导致没有被吃掉的字符集，buf\_size用于指示其位置

currString数组用于存放标识符或数字或者是导致ERROR的字符，name\_pos用于指示currString数组最长的长度

fp则是打开的文件指针

**getNext函数：**

int getNext() {

    int c;

    if (buf\_size == 0) {

        c = getc(fp);

    }

    else {

        c = buffer[0];

        int i;

        for (i = 0; i < buf\_size; i++) {

            buffer[i] = buffer[i + 1];

        }

        buf\_size--;

    }

    return c;

}

这一部分进行字符的读取，使用buffer数组处理回溯的情况（详见下一个函数backtrace），如果buffer数组为空，那么从打开的文件中读取下一个元素，否则，读取buffer数组的第一个元素，之后将buffer数组整体前移，大小减1。

**backtrace函数：**

void backtrace(char *c*) {

    buffer[buf\_size++] = *c*;

}

这一部分具体的处理回溯的阶段，如果有回溯发生，那么就将没有被吃进的字符放入buffer数组，并且buffer数组的大小加1，读取时buffer数组的优先级要优于从文件读取。

**getToken函数：**

TokenType getToken() {

    StateType state = START;

    TokenType currToken;

    int save = 0;

    while(state != DONE) {

        int c = getNext();

        switch(state) {

            case START: //位于START状态

            if (c == ' ' || c == '\t' || c == '\n') {

                state = START;

            }

            else if (c == '{') {

                state = INCOMMENT;

            }

            else if (isdigit(c)) {

                currString[name\_pos++] = c;

                save = 1;

                state = INNUM;

            }

            else if (isalpha(c)) {

                currString[name\_pos++] = c;

                save = 1;

                state = INID;

            }

            else if (c == ':') {

                currString[name\_pos++] = c;

                state = INASSIGN;

            }

            else {

                state = DONE;

                switch(c) {

                    case EOF:

                    currToken = ENDFILE;

                    break;

                    case '+':

                    currToken = PLUS;

                    break;

                    case '-':

                    currToken = MINUS;

                    break;

                    case '\*':

                    currToken = TIMES;

                    break;

                    case '/':

                    currToken = DIVIDE;

                    break;

                    case '=':

                    currToken = EQULE;

                    break;

                    case '<':

                    currToken = LE;

                    break;

                    case '(':

                    currToken = LB;

                    break;

                    case ')':

                    currToken = RB;

                    break;

                    case ';':

                    currToken = SEMI;

                    break;

                    default:

                    currString[name\_pos++] = c;

                    currToken = ERROR;

                }

            }

            break;

            case INNUM://位于INNUM状态

            if (!isdigit(c)) {

                backtrace(c);

                state = DONE;

                currToken = NUM;

            }

            else currString[name\_pos++] = c;

            break;

            case INID://位于INID状态

            if (!isalpha(c)) {

                backtrace(c);

                state = DONE;

                currToken = ID;

            }

            else currString[name\_pos++] = c;

            break;

            case INASSIGN://位于INASSIGN状态

            if (c == '=') {

                state = DONE;

                currToken = ASSIGN;

            }

            else {

                backtrace(c);

                state = DONE;

                currToken = ERROR;

            }

            break;

            case INCOMMENT://位于INCOMMENT状态

            if (c == EOF) {

                state = DONE;

                currToken = ENDFILE;

            }

            else if (c == '}') {

                state = START;

            }

            break;

            case DONE://结束状态

            break;

            default:

            state = DONE;

            currString[name\_pos++] = c;

            currToken = ERROR;

        }

        if (state == DONE) { // 查找ID是否是保留字

            currString[name\_pos] = '\0';

            if (currToken == ID) {

                currToken = getRealToken();

            }

            name\_pos = 0;

        }

    }

    return currToken;

}

这一部分则是根据当前状态和下一个输入来进行状态的转移。首先当状态不是DONE时，要调用读取函数对下一个字符进行读取。

在START状态即初始状态，具体的状态转移的操作如上面DFA所示，细节部分：在转移到INNUM和INID时，优于最终要输出对应的NUM值或ID值，所以要把当前字符放到currString中，转移到INASSIGN时也进行这样的操作，由于可能发生的错误是:?，此时需要把:加入到currString中，为了之后进行错误输出；另外一种情况就是读取的字符为特殊符号，这时候会直接转移到DONE状态，这是也要进行一个选择，判断这个特殊符号是什么，并将对应的TokenType赋值给currToken；剩余不符合的情况也需要加入到currString中，作为错误输出。

在INNUM和INID状态，同样的，如果符合条件则继续加入到currString中，否则进行回溯，转移到DONE，设置currToken为对应的Token。

INASSIGN内，则只需要接收到’=’就转移到DONE即可，其余情况回溯并currToken=ERROR。

最后的INCOMMENT内则是接收到’}’或者EOF就转移，否则继续。

当达到DONE后，由于可能的ID为保留字，还需要进行判断。

getRealToken函数：

TokenType getRealToken() {

    int i;

    for (i = 0; i < 8; i++) {

        if (strcmp(currString, reserved[i].str) == 0) {

            return reserved[i].token;

        }

    }

    return ID;

}

直接在之前定义的保留字集中查找即可，找到了则是保留字，否则就为ID

printResult函数：

void printResult(TokenType *curr*) {

*// printf("%d\t", line\_no);*

    switch(*curr*) {

        case IF:

        printf("reserved word: if\n");

        break;

        case THEN:

        printf("reserved word: then\n");

        break;

        case ELSE:

        printf("reserved word: else\n");

        break;

        case END:

        printf("reserved word: end\n");

        break;

        case REPEAT:

        printf("reserved word: repeat\n");

        break;

        case UNTIL:

        printf("reserved word: until\n");

        break;

        case READ:

        printf("reserved word: read\n");

        break;

        case WRITE:

        printf("reserved word: write\n");

        break;

        case ASSIGN:

        printf(":=\n");

        break;

        case EQULE:

        printf("=\n");

        break;

        case LE:

        printf("<\n");

        break;

        case PLUS:

        printf("+\n");

        break;

        case MINUS:

        printf("-\n");

        break;

        case TIMES:

        printf("\*\n");

        break;

        case DIVIDE:

        printf("/\n");

        break;

        case LB:

        printf("(\n");

        break;

        case RB:

        printf(")\n");

        break;

        case SEMI:

        printf(";\n");

        break;

        case NUM:

        printf("NUM, val= ");

        printf("%s\n", currString);

        currString[name\_pos] = '\0';

        break;

        case ID:

        printf("ID, name= ");

        printf("%s\n", currString);

        currString[name\_pos] = '\0';

        break;

        case ENDFILE:

        printf("EOF\n");

        break;

        case ERROR:

        printf("Error: %s\n", currString);

        break;

        default:

        printf("Error\n");

    }

}

就是一个按照TokenType输出的函数，没什么细节。

1. 样例测试
   1. 正例

{ Sample program

in TINY language -

computes factorial

}

read x; { input an integer }

if 0 < x then { don't compute if x <= 0 }

fact := 1;

repeat

fact := fact \* (x / 3);

x := x - 1 + 0

until x = 0;

write fact { output factorial of x }

else write fact

end

**输出：**

reserved word: read

ID, name= x

;

reserved word: if

NUM, val= 0

<

ID, name= x

reserved word: then

ID, name= fact

:=

NUM, val= 1

;

reserved word: repeat

ID, name= fact

:=

ID, name= fact

\*

(

ID, name= x

/

NUM, val= 3

)

;

ID, name= x

:=

ID, name= x

-

NUM, val= 1

+

NUM, val= 0

reserved word: until

ID, name= x

=

NUM, val= 0

;

reserved word: write

ID, name= fact

reserved word: else

reserved word: write

ID, name= fact

reserved word: end

EOF

* 1. 反例1

{

if x = fact then

end

}

read x;

**\\**

str := **"**string**"**

str **:-** 2;

w ww

str123

**输出：**

reserved word: read

ID, name= x

;

Error: \

Error: \

ID, name= str

:=

Error: "

ID, name= string

Error: "

ID, name= str

Error: :

-

NUM, val= 2

;

ID, name= w

ID, name= ww

ID, name= str

NUM, val= 123

EOF

* 1. 反例2

{

if x = fact then

end

{This is Error}

**}**

**输出：**

Error: }

EOF

以上样例测试均与预期相符。