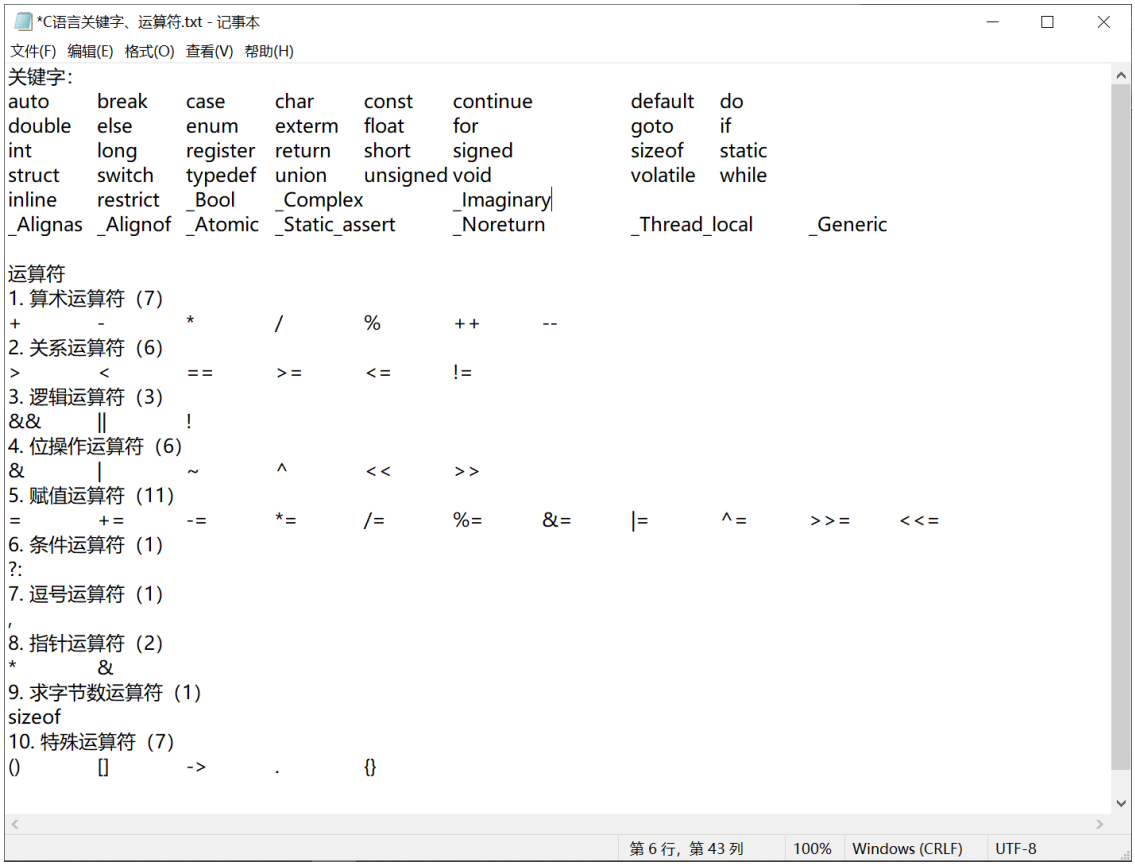
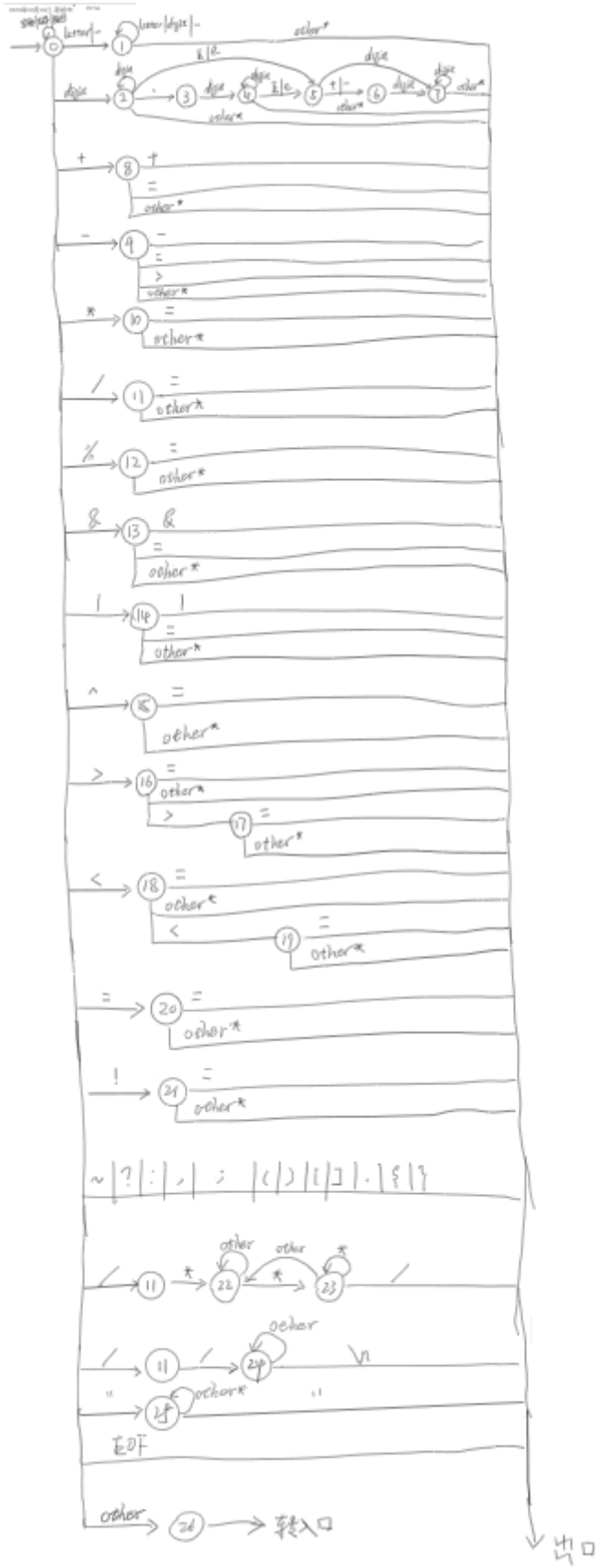
实验报告

自思路

1. 将C语言的关键字和运算符列出来



2. 根据关键字和运算符画出自动机



3. 根据自动机写C语言代码

源码

```
#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>
#include<stdlib.h>
```

```
#include<string.h>

#define KEYWORD_NUMBER 44
#define BUFFERLENGTH 128

//宏函数，填充缓冲区，参数为左或右缓冲区的首地址
//#define FILLBUFFER(a) {fread(a,BUFFERLENGTH/2-1,1,fp_in);a[BUFFERLENGTH/2-
1]=EOF;}
#define FORWARD_PTR_INC \
    forward_ptr++;\
    char_num++;\
    if(*forward_ptr == EOF)\
    {\
        if (forward_ptr == &left[BUFFERLENGTH / 2 - 1])\
        {\
            FillBuffer(fp_in,right);\
            forward_ptr++;\
        }\
        else if(forward_ptr == &right[BUFFERLENGTH / 2 - 1])\
        {\
            FillBuffer(fp_in,left);\
            forward_ptr = left;\
        }\
    }

typedef unsigned bool;

//构造关键字表
int CreateKeywordsTable(char** keywords);

//填充缓冲区
void FillBuffer(FILE * fp, char* buffer);

//词法分析
int LexicalAnalysis(char** keywords);

//文件输出
void FileOut(FILE * fp, char* token, char* attribute);

//错误处理
void Error(FILE* fp,int line);

//判断是否是字符
bool IsLetter(char c);

//判断是否是数字
bool IsDigit(char c);

//提取记号
int GetAttribute(char *buffer, char* begin, char* forward, char* attribute);

//判断保留字是不是关键字
bool IsKey(char** keywords, char* attribute);

int main(void)
{
    int ret = 0;
    int i = 0;
```

```
    char* keywords[KEYWORD_NUMBER];

    //构造关键字表
    ret = CreateKeywordsTable(keywords);

    for (i = 0; i < KEYWORD_NUMBER; i++)
    {
        printf("%s\n", keywords[i]);
    }

    //词法分析
    ret = LexicalAnalysis(keywords);

    return ret;

}


//构造关键字表
int CreateKeywordsTable(char** keywords)
{
    int ret = 0;

    if (keywords == NULL)
    {
        ret = -1;
        printf("func CreateKeywordsTable() err: %d(keywords == NULL)\n", ret);
        return ret;
    }

    keywords[0] = "auto";
    keywords[1] = "break";
    keywords[2] = "case";
    keywords[3] = "char";
    keywords[4] = "const";
    keywords[5] = "continue";
    keywords[6] = "default";
    keywords[7] = "do";
    keywords[8] = "double";
    keywords[9] = "else";
    keywords[10] = "enum";
    keywords[11] = "exterm";
    keywords[12] = "float";
    keywords[13] = "for";
    keywords[14] = "goto";
    keywords[15] = "if";
    keywords[16] = "int";
    keywords[17] = "long";
    keywords[18] = "register";
    keywords[19] = "return";
    keywords[20] = "short";
    keywords[21] = "signed";
    keywords[22] = "sizeof";
    keywords[23] = "static";
    keywords[24] = "struct";
    keywords[25] = "switch";
    keywords[26] = "typedef";
    keywords[27] = "union";
```

```
    keywords[28] = "unsigned";
    keywords[29] = "void";
    keywords[30] = "volatile";
    keywords[31] = "while";
    keywords[32] = "inline";
    keywords[33] = "restrict";
    keywords[34] = "_Bool";
    keywords[35] = "_Complex";
    keywords[36] = "_Imaginary";
    keywords[37] = "_Alignas";
    keywords[38] = "_Alignof";
    keywords[39] = "_Atomic";
    keywords[40] = "_Static_assert";
    keywords[41] = "_Noreturn";
    keywords[42] = "_Thread_local";
    keywords[43] = "_Generic";

    return ret;
}

//文件输出
void FileOut(FILE* fp,char* token, char* attribute)
{
    fprintf(fp, "<%s,%s> ", token, attribute);
}

//填充缓冲区
void FillBuffer(FILE* fp, char* buffer)
{
    char ch = fgetc(fp);
    int i = 0;
    while (i < (BUFFERLENGTH / 2 - 1) && ch != EOF)
    {
        buffer[i] = ch;
        ch = fgetc(fp);
        i++;
    }
    buffer[i] = EOF;
    buffer[(BUFFERLENGTH / 2 - 1)] = EOF;

    i = 0;
    while (buffer[i] != EOF)
    {
        printf("%c", buffer[i]);
        i++;
    }
}


//错误处理
void Error(FILE* fp, int line)
{
    fprintf(fp, "第%d行：无法识别的标识符\n", line);
}

//判断是否是字符
bool IsLetter(char c)
{
```

```
    if ((c <= 'z' && c >= 'a') || (c <= 'Z' && c >= 'A'))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

//判断是否是数字
bool IsDigit(char c)
{
    if (c <= '9' && c >= '0')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

//提取记号
int GetAttribute(char* buffer, char* begin, char* forward, char* attribute)
{
    char temp[128] = { 0 };
    char* ch = NULL;
    //开始指针在向前指针后面的情况
    if (forward >= begin)
    {
        //开始指针在左缓冲区，向前指针在右缓冲区的情况
        if ((forward >= &buffer[BUFFERLENGTH / 2 - 1]) && (begin <
&buffer[BUFFERLENGTH / 2 - 1]))
        {
            ch = strchr(begin, EOF);
            strncpy(attribute, begin, (size_t)(ch - begin));
            attribute[(size_t)(ch - begin)] = '\0';

            strncpy(temp, ch + 1, (size_t)(forward - ch + 1));
            temp[(size_t)(forward - ch + 1)] = '\0';

            strcat(attribute, temp);
        }
        //开始指针和向前指针在同一个缓冲区
        else
        {
            strncpy(attribute, begin, (size_t)(forward - begin));
            attribute[(size_t)(forward - begin)] = '\0';
        }
    }
    //开始指针在向前指针前面的情况，及开始指针在右缓冲区，向前指针在左缓冲区
    else
    {
        ch = strchr(begin, EOF);
        strncpy(attribute, begin, (size_t)(ch - begin));
        attribute[(size_t)(ch - begin)] = '\0';
```

```
        strncpy(temp, buffer, (size_t)(forward - buffer));
        temp[(size_t)(forward - buffer)] = '\0';
        strcat(attribute, temp);
    }
}

//判断保留字是不是关键字
bool IsKey(char** keywords, char* attribute)
{
    int i = 0;

    for (i = 0; i < KEYWORD_NUMBER; i++)
    {
        if (strcmp(keywords[i], attribute) == 0)
        {
            return 1;
        }
    }

    return 0;
}

//词法分析
int LexicalAnalysis(char** keywords)
{
    int ret = 0;

    int state = 0;
    int i = 0;

    FILE* fp_in = NULL;
    FILE* fp_out = NULL;
    FILE* fp_error = NULL;
    char buffer[BUFFERLENGTH] = {0};
    char* left = buffer;
    char* right = &(buffer[BUFFERLENGTH / 2]);

    char ch = '0';

    char* begin_ptr = NULL;
    char* forward_ptr = NULL;

    int linefeeds = 0;
    int char_num = 0;
    char attribute[128] = { 0 };

    //打开输入文件
    fp_in = fopen("files/in.txt", "r");
    if (fp_in == NULL)
    {
        ret = -1;
        printf("func fopen() err: %d(fp_in == NULL)\n", ret);
        return ret;
    }

    //打开输入文件
    fp_out = fopen("files/out.txt", "w");
    if (fp_out == NULL)
```

```
    {
        ret = -1;
        printf("func fopen() err: %d(fp_out == NULL)\n", ret);
        return ret;
    }

    //打开输入文件
    fp_error = fopen("files/error.txt", "w");
    if (fp_out == NULL)
    {
        ret = -1;
        printf("func fopen() err: %d(fp_out == NULL)\n", ret);
        return ret;
    }

    FillBuffer(fp_in,left);

    printf("left:\n%s\n", left);

    state = 0;
    begin_ptr = left;
    forward_ptr = left;

    do
    {
        switch (state)
        {
        case 0:
            switch (*forward_ptr)
            {
            case 'a':
            case 'b':
            case 'c':
            case 'd':
            case 'e':
            case 'f':
            case 'g':
            case 'h':
            case 'i':
            case 'j':
            case 'k':
            case 'l':
            case 'm':
            case 'n':
            case 'o':
            case 'p':
            case 'q':
            case 'r':
            case 's':
            case 't':
            case 'u':
            case 'v':
            case 'w':
            case 'x':
            case 'y':
            case 'z':
            case 'A':
            case 'B':
```

```
            case 'C':
            case 'D':
            case 'E':
            case 'F':
            case 'G':
            case 'H':
            case 'I':
            case 'J':
            case 'K':
            case 'L':
            case 'M':
            case 'N':
            case 'O':
            case 'P':
            case 'Q':
            case 'R':
            case 'S':
            case 'T':
            case 'U':
            case 'V':
            case 'W':
            case 'X':
            case 'Y':
            case 'Z':
            case '_':
                state = 1;
                FORWARD_PTR_INC;
                break;
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                state = 2;
                FORWARD_PTR_INC;
                break;
            case '+':
                state = 8;
                FORWARD_PTR_INC;
                break;
            case '-':
                state = 9;
                FORWARD_PTR_INC;
                break;
            case '*':
                state = 10;
                FORWARD_PTR_INC;
                break;
            case '/':
                state = 11;
                FORWARD_PTR_INC;
                break;
            case '%':
```

```
                state = 12;
                FORWARD_PTR_INC;
                break;
            case '&':
                state = 13;
                FORWARD_PTR_INC;
                break;
            case '|':
                state = 14;
                FORWARD_PTR_INC;
                break;
            case '^':
                state = 15;
                FORWARD_PTR_INC;
                break;
            case '>':
                state = 16;
                FORWARD_PTR_INC;
                break;
            case '<':
                state = 18;
                FORWARD_PTR_INC;
                break;
            case '=':
                state = 20;
                FORWARD_PTR_INC;
                break;
            case '!':
                state = 21;
                FORWARD_PTR_INC;
                break;
            case '~':FileOut(fp_out, "~", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '?':FileOut(fp_out, "?", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case ':':FileOut(fp_out, ":", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case ',':FileOut(fp_out, ",", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '(':FileOut(fp_out, "(", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case ')':FileOut(fp_out, ")", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '[':FileOut(fp_out, "[", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case ']':FileOut(fp_out, "]", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '.':FileOut(fp_out, ".", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '{':FileOut(fp_out, "{", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '}':FileOut(fp_out, "}", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case ';':FileOut(fp_out, ";", "-"); FORWARD_PTR_INC; begin_ptr =
forward_ptr;state = 0; break;
            case '"':
                state = 25;
                FORWARD_PTR_INC;
```

```
                break;
            case ' ':
            case '\t':
                state = 0;
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                break;
            case '\n':
                fprintf(fp_out, "\n");
                state = 0;
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                linefeeds++;
                break;
            case EOF:
                break;
            default:
                Error(fp_error,linefeeds);
                state = 0;
                break;
            };
            break;
        case 1:
            if (IsLetter(*forward_ptr) || IsDigit(*forward_ptr) || *forward_ptr
== '_')
            {
                state = 1;
                FORWARD_PTR_INC;
            }
            else
            {
                GetAttribute(buffer, begin_ptr, forward_ptr, attribute);
                //检查标识符是不是保留字
                if (IsKey(keywords, attribute))
                {
                    FileOut(fp_out, "keyword", attribute);
                }
                else
                {
                    FileOut(fp_out, "id", attribute);
                }
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 2:
            if (IsDigit(*forward_ptr))
            {
                state = 2;
                FORWARD_PTR_INC;
            }
            else if ((*forward_ptr == 'E') || (*forward_ptr == 'e'))
            {
                state = 5;
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '.')
            {
```

```
                state = 3;
                FORWARD_PTR_INC;
            }
            else
            {
                GetAttribute(buffer, begin_ptr, forward_ptr, attribute);
                FileOut(fp_out, "Num", attribute);
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 3:
            if (IsDigit(*forward_ptr))
            {
                state = 4;
                FORWARD_PTR_INC;
            }
            else
            {
                Error(fp_error,linefeeds);
                state = 0;
            }
            break;
        case 4:
            if (IsDigit(*forward_ptr))
            {
                state = 4;
                FORWARD_PTR_INC;
            }
            else if ((*forward_ptr == 'E') || (*forward_ptr == 'e'))
            {
                state = 5;
                FORWARD_PTR_INC;
            }
            else
            {
                GetAttribute(buffer, begin_ptr, forward_ptr, attribute);
                FileOut(fp_out, "Num", attribute);
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 5:
            if (IsDigit(*forward_ptr))
            {
                state = 7;
                FORWARD_PTR_INC;
            }
            else if ((*forward_ptr) == '+' || (*forward_ptr) == '-')
            {
                state = 6;
                FORWARD_PTR_INC;
            }
            else
            {
                Error(fp_error,linefeeds);
                state = 0;
            }
```

```
            break;
        case 6:
            if (IsDigit(*forward_ptr))
            {
                state = 7;
                FORWARD_PTR_INC;
            }
            else
            {
                Error(fp_error,linefeeds);
                state = 0;
            }
            break;
        case 7:
            if (IsDigit(*forward_ptr))
            {
                state = 7;
                FORWARD_PTR_INC;
            }
            else
            {
                GetAttribute(buffer, begin_ptr, forward_ptr, attribute);
                FileOut(fp_out, "Num", attribute);
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 8:
            if (*forward_ptr == '+')
            {
                FileOut(fp_out, "++", "-");
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '=')
            {
                FileOut(fp_out, "+=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "+", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 9:
            if (*forward_ptr == '-')
            {
                FileOut(fp_out, "--", "-");
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '=')
            {
                FileOut(fp_out, "-=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
```

```
                FileOut(fp_out, "-", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 10:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "*=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "*", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 11:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "/=", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else if (*forward_ptr == '*')
            {
                state = 22;
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '/')
            {
                state = 24;
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "/", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 12:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "%=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "%", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 13:
            if (*forward_ptr == '&')
```

```
            {
                FileOut(fp_out, "&&", "-");
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '=')
            {
                FileOut(fp_out, "&=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "&", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 14:
            if (*forward_ptr == '|')
            {
                FileOut(fp_out, "||", "-");
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '=')
            {
                FileOut(fp_out, "|=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "|", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 15:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "^=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "^", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 16:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, ">=", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else if (*forward_ptr == '>')
            {
                state = 17;
```

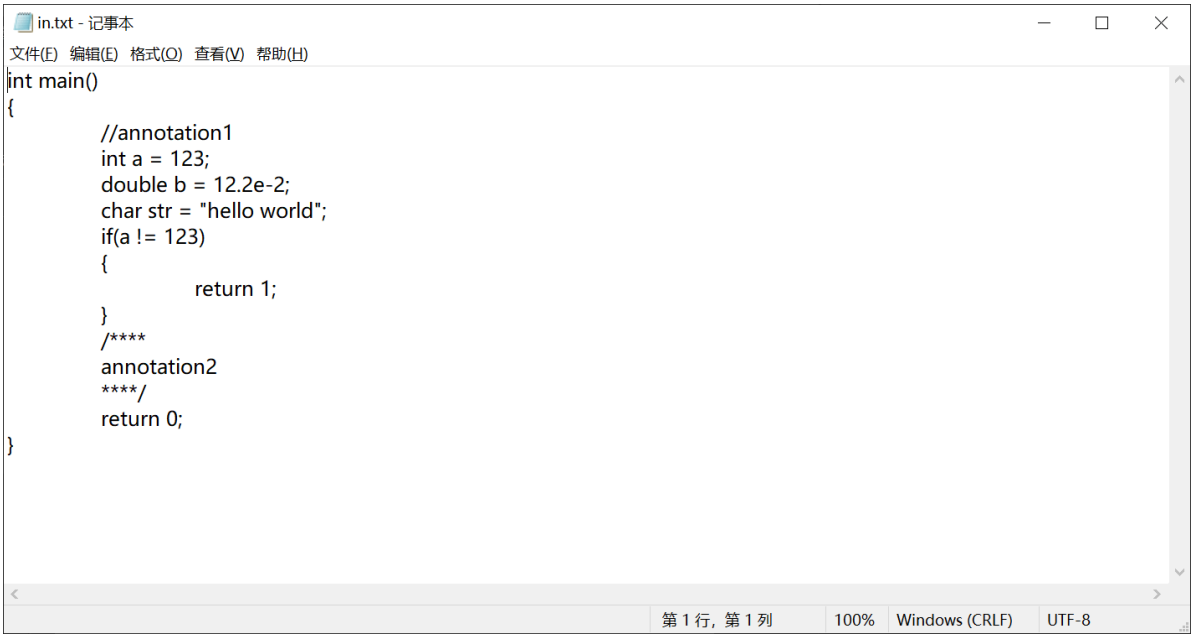
```
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, ">", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 17:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, ">>=", "-");
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, ">>", "-");
            }
            begin_ptr = forward_ptr;
            state = 0;
            break;
        case 18:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "<=", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else if (*forward_ptr == '<')
            {
                state = 19;
                FORWARD_PTR_INC;
            }
            else
            {
                FileOut(fp_out, "<", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 19:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "<<=", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                FileOut(fp_out, "<<", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 20:
```

```
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "==", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                FileOut(fp_out, "=", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 21:
            if (*forward_ptr == '=')
            {
                FileOut(fp_out, "!=", "-");
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                FileOut(fp_out, "!", "-");
                begin_ptr = forward_ptr;
                state = 0;
            }
            break;
        case 22:
            if (*forward_ptr == '*')
            {
                state = 23;
                FORWARD_PTR_INC;
            }
            else
            {
                state = 22;
                FORWARD_PTR_INC;
            }
            break;
        case 23:
            if (*forward_ptr == '*')
            {
                state = 23;
                FORWARD_PTR_INC;
            }
            else if (*forward_ptr == '/')
            {
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                state = 22;
                FORWARD_PTR_INC;
            }
```

```
            break;
        case 24:
            if (*forward_ptr == '\n')
            {
                linefeeds++;
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                state = 24;
                FORWARD_PTR_INC;
            }
            break;
        case 25:
            if (*forward_ptr == '\"')
            {
                GetAttribute(buffer, begin_ptr, forward_ptr+1, attribute);
                FileOut(fp_out, "String", attribute);
                FORWARD_PTR_INC;
                begin_ptr = forward_ptr;
                state = 0;
            }
            else
            {
                state = 25;
                FORWARD_PTR_INC;
            }
            break;
        default:
            Error(fp_error,linefeeds);
            state = 0;
            break;
        };
    }while (*forward_ptr != EOF);

    fprintf(fp_out, "\n行数：%d\n", linefeeds+1);
    fprintf(fp_out, "\n字符数：%d\n", char_num);



    //关闭文件
    if (fp_in != NULL)
    {
        fclose(fp_in);
        fp_in = NULL;
    }
    if (fp_out != NULL)
    {
        fclose(fp_out);
        fp_out = NULL;
    }
    if (fp_error != NULL)
    {
        fclose(fp_error);
        fp_error = NULL;
    }
```

```
    return ret;
}
```

输入



输出