苏 州 大 学 实 验 报 告

院、系	计算机科学与技术学院	年级专业	21 计科	姓名	赵鹏	学号	2127405037
课程名称	编译原理实践					成绩	
指导教师	段湘煜	同组实验者		实验日期	2023.8.28		

实 验 名 称 TEST 语言的词法分析

一. 实验题目

设计与实现 TEST 语言的词法分析器

TEST 语言介绍如下：TEST 语言所有变量都是整型变量，具有 if、while、for 等控制语句。注释用/*和*/括起来，但注释不能嵌套。TEST 语言的表达式局限于布尔表达式和算数表达式。

TEST 语言的单词符号有：

标识符：字母开头,后接字母或数字。

保留字(它是标识符的子集)：if、else、for、 while、do、int。

无符号整数：由 0~9 数字组成。

分界符：如+、-、*、/、(、)、;、，等单分界符；还有双字符分界符>=、<=、!=、 ==等。

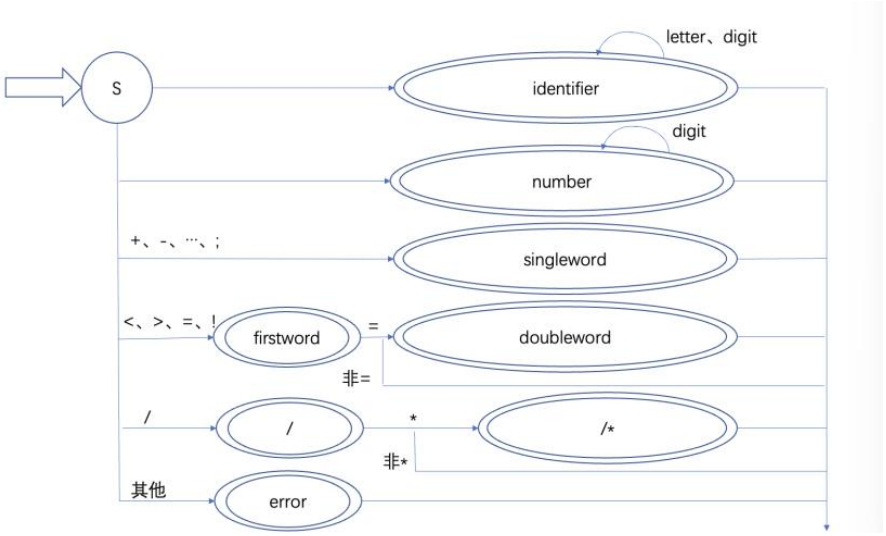
注释符：用/*和*/括起来。 词法分析程序并不输出注释,在词法分析阶段注释的内容将被删掉。为了从 源程序字符流中正确识别出各类单词符号，相邻的标识符整数或保留字之间至少 要用一个空格分开。

二. 实验原理及流程框图

TEST 语言的各类单词的符号的正则表达式如下：

$$
\begin{aligned}
&< identifier > -> < letter > | < identifier >< letter > | < identifier >< digit >\\
&< number > ->< digit > | < number >< digit >\\
&< letter > -> a|b|..|z|A|B|..|Z\\
&< digit > -> 1|2| ...|9|0\\
&< singleword > -> +| - | * |/| = |(|)|{|}|:|,|;< | > |!\\
&< doubleword > ->>= | <= |! = | ==\\
&< comment\ first > ->/*\\
&< commentlast > ->*/
\end{aligned}
$$

根据上述表达式可以得到下图的词法分析的状态图：



由于 TEST 语言语法规则相对简单，故在实现词法分析时为了方便，可以采用依次读入符号，模拟词法分析的过程的办法，通过读入的字符判断接下来的单词符号是那种类型或进行错误处理，并进行后续处理，程序的流程框图如下图所示:

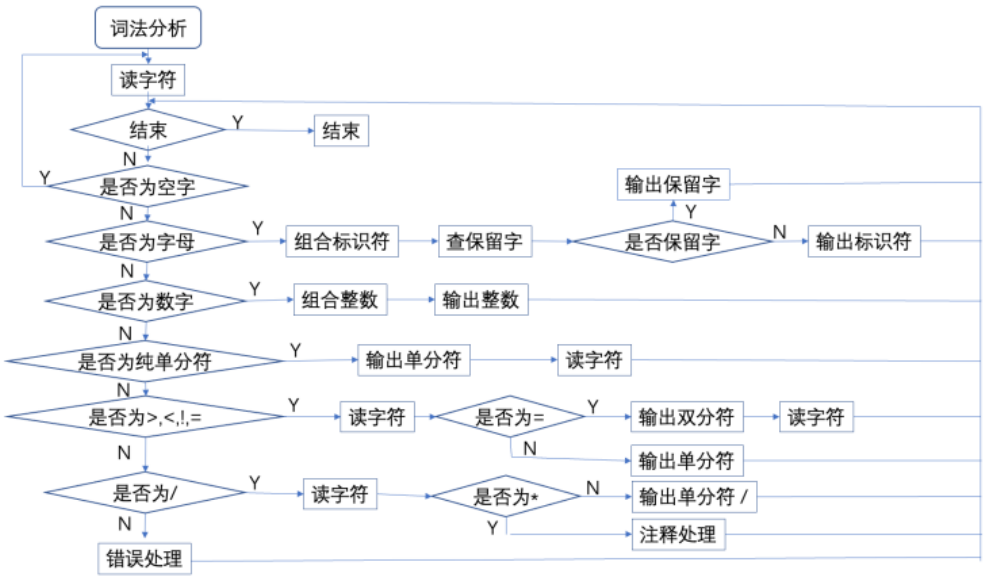


图 3. 词法分析程序流程图

三. 实验步骤

1. 首先定义 Lexer 类，主要功能是从输入文件中读取字符并将其识别为不同类型的词法单元，然后将识别的结果写入输出文件。

   Lexer 类的成员变量包括输入文件流 inputFile,输出文件流 outputFile、当前行号 line、当前列号 col 以及当前字符 ch。

   Lexer 类的构造函数接受两个参数作为输入文件名和输出文件名并打开，为词法分析做准备。析构函数在 Lexer 对象被销毁时关闭文件。

   Lexer 类的 run()方法作为词法分析的主程序，run()方法不断从输入文件中读取字符，根据字符的类型进行不同调用不同的私有方法进行处理，详细过程如词法分析程序流程图所示。在处理过程中将识别到的词法单元写入输出文件。

Lexer 类还定义类多个私有方法，用于处理不同词法单元，包括空白字符、标识符、数字、单双字符，每个方法有 0、1、2 三种返回值，0 代表文件结束，未出问题，词法分析成功。1 代表出现错误，词法分析结束，2 代表未出现问题，继续进行词法分析。

2. 在主函数中输入输入文件和输出文件名，并构造 Lexer 对象，调用构造对象的 run 函数进行词法分析，run()方法返回 0 或 1 代表分析结果，1 代表存在错误，0 代表成功完成词法分析。错误位置及错误信息都将同时输出在文件和在终端里。

四. 实验结果及分析

测试一：

输入：

```
{
int i;
int n;
int j;
j=1;
read n;
for(i=1; i<=n; i=i+1)
j=j*i;
write j;
}
```

输出：

```
{ {
int int
ID i
; ;
int int
ID n
; ;
int int
ID j
; ;
ID j
= =
NUM 1
; ;
read read
ID n
; ;
for for
( (
ID i
= =
NUM 1
```

; ;
ID i
<= <=
ID n
; ;
ID i
= =
ID i
+ +
NUM 1
))
ID j
= =
ID j
* *
ID i
; ;
write write
ID j
; ;
} }

测试二：

输入：

```
{
    int a;
    a=0000012345;
    {
        int b = 10;
    }
    a=123;
}
```

输出：

{ {
int int
ID a
; ;
ID a
= =
NUM 0000012345
; ;
{ {

int int
ID b
= =
NUM 10
; ;
} }
ID a
= =
NUM 123
; ;
} }

测试三：

输入：

int a;
a = 123456;
int 1c;

输出：

int int
ID a
; ;
ID a
= =
NUM 123456
; ;
int int
[Error] Invalid token! line:3 col:6

测试四：

输入：

{
int a ;
a = 123;
/*
}

输出：

{ {
int int
ID a
; ;
ID a
= =

NUM 123
;;
[Error] Unclosed comment,line:6 col:0

分析：代码能够正确的进行词法分析，对于不符合规则的情况，如：注释不匹配等问题都能够正确处理。

五. 实验总结

通过本次实验，我对 TEST 语言的词法分析有了一定的了解，并能够编写代码实现 TEST 语言的词法分析。尽管在编写代码过程中出现了许多小问题，经过不断的分析思考，我解决了遇到的各种问题。在代码编写完后的测试阶段，我也测试出了自己代码的很多问题，经过调试，各种问题也基本解决。我对词法分析的过程有了更加深入的理解。

六. 代码

```
 #include <iostream>
#include <fstream>
#include <string>
#include <vector>

class Lexer
{
private:
    std::string keywords[10] = {"if", "else", "for", "while", "do", "int", "read",
"write"};
    const std::string singleSymbols = "+-*(){};,:";
    const std::string doubleSymbols = "><=!";

    std::ifstream inputFile;
    std::ofstream outputFile;
    int line;
    int col;
    char ch;
    bool inComment;

public:
    Lexer(const std::string &inputFileName, const std::string &outputFileName) : line(1),
col(1)
    {
        inputFile.open(inputFileName);
        inputFile.unsetf(std::ios_base::skipws);
        outputFile.open(outputFileName);
        ch = char();
    }

    ~Lexer()
```

```
{
    inputFile.close();
    outputFile.close();
}

int run()// 0:正常结束  1:有问题  2：继续读
{
    inputFile.get(ch);
    while (1)
    {

        if (ch == EOF)
            return 0;

        if (std::isspace(ch))
        {
            int ret = handleWhitespace();

            if(ret==0)
                return 0;
            else if (ret==1)
                return 1;


        }
        else if (std::isalpha(ch))
        {
            int ret = handleIdentifier();
            if(ret==0)
                return 0;
            else if (ret==1)
                return 1;
        }
        else if (std::isdigit(ch))
        {
            int ret = handleNumber();
            if(ret==0)
                return 0;
            else if (ret==1)
                return 1;
        }
        else if (singleSymbols.find(ch) != std::string::npos)
        {
```

```
            int ret = handleSingleSymbol();
               if(ret==0)
                   return 0;
               else if (ret==1)
                   return 1;
           }
           else if (doubleSymbols.find(ch) != std::string::npos)
           {
               int ret = handleDoubleSymbol();
               if(ret==0)
                   return 0;
               else if (ret==1)
                   return 1;

           }
           else if (ch == '/')
           {
               int ret = handleComment();
               if(ret==0)
                   return 0;
               else if (ret==1)
                   return 1;
           }
           else
           {
               handleInvalidToken();
               return 1;
           }
       }
   }

private:
// 0:正常结束  1:有问题  2：继续读
   int handleWhitespace()
   {
       if (ch == '\n')
       {
           line++;
           col = 0;
       }
       col++;
       if (!inputFile.get(ch))
       {
           if(inComment)
```

```
        {
            outputFile << "[Error] Unclosed comment" << "\n";
            std::cout<< "[Error] Unclosed comment" << "\n";
            return 1;
        }
        return 0;
    }

    return 2;
}

int handleIdentifier()
{

    std::string temp;

    while (std::isalnum(ch))
    {
        temp.push_back(ch);
        col++;

        if (!inputFile.get(ch))
        {

        handleID(temp);
        if(inComment)
        {
            outputFile << "[Error] Unclosed comment" << "\n";
            std::cout<< "[Error] Unclosed comment" << "\n";
            return 1;
        }

            return 0;
        }
    }


    handleID(temp);

    return 2;
}

int handleNumber()
{
```

```
        std::string temp;

        while (std::isdigit(ch))
        {
            temp.push_back(ch);
            if (!inputFile.get(ch))
            {
            if(inComment)
            {
                 outputFile << "NUM " << temp << "\n";
                outputFile << "[Error] Unclosed comment" << "\n";
                std::cout<< "[Error] Unclosed comment" << "\n";
                return 1;
            }

                return 0;
            }
            col++;
        }

        if (!isdigit(ch))
        {
            if (ch == ';')
            {
            outputFile << "NUM " << temp << "\n";
            return 2;
            }
            else
            {
              outputFile << "[Error] Invalid token! line:" << line << " col:" << col <<
"\n";
              std::cout<< "[Error] Invalid token! line:" << line << " col:" << col << "\n";
              return 1;
            }
        }

        return 2;
    }

    int handleSingleSymbol()
    {
        char pre = ch;
        outputFile << ch << " " << ch << "\n";
```


教务处制

```
        if (!inputFile.get(ch))
        {
            if(inComment)
            {
                outputFile << "[Error] Unclosed comment"<< "\n";
                std::cout<< "[Error] Unclosed comment" << "\n";
                return 1;
            }
            return 0;

        }
        else
        {   if(pre=='*'&&ch=='/')
               {
                outputFile <<"[Error] Redundant right comment."<< line << " col:" << col <<
"\n";
                std::cout<<"[Error] Redundant right comment."<< line << " col:" << col <<
"\n";
                return 1;
                }
            return 2;
        }
    }

    int handleDoubleSymbol()
    {
        char pre, nxt;
        pre = ch;
        if (!inputFile.get(nxt))
        {
            if(inComment)
            {
                outputFile << "[Error] Unclosed comment" <<  "\n";
                std::cout<< "[Error] Unclosed comment" << "\n";
                return 1;
            }
            outputFile << pre << " " << pre << "\n";
            return 0;
        }
        col++;

        if (doubleSymbols.find(nxt) != std::string::npos)
        {
```

```
            outputFile << pre << nxt << " " << pre << nxt << "\n";

            if (!inputFile.get(ch))
            {

                outputFile << "[Error] Invalid token! line:" << line << " col:" << col <<
"\n";
                std::cout<< "[Error] Invalid token! line:" << line << " col:" << col <<
"\n";
                return 1;
            }
            else
            {

                return 2;
            }
        }
        else
        {
            outputFile << pre << " " << pre << "\n";
            ch = nxt;
            return 2;
        }
        return 2;
    }

    int handleComment()
    {

        char pre, nxt;
        pre = ch;
        if (!inputFile.get(nxt))
        {
            outputFile << "/ /\n";
            return 0;
        }
        col++;
        if (nxt == '*')
        {
            inComment = 1;
            if (!inputFile.get(pre))
            {
                        outputFile << "[Error] Unclosed comment" <<  "\n";
                std::cout<< "[Error] Unclosed comment" << "\n";
```

```
            return 1;
        }
        col++;

        while (inputFile.get(nxt))
        {
            if(nxt=='\n')
            {
                line++;
                col = 0;
            }
            else
            {
                col++;
            }
            if (pre == '*' && nxt == '/')
                {
                inComment = 0;
                break;
                }
            pre = nxt;
        }

        if(!inputFile.get(ch))
        {

        if(inComment)
        {
          outputFile << "[Error] Unclosed comment" <<  "\n";
            std::cout<< "[Error] Unclosed comment" << "\n";
            return 1;
        }
            return 0;
        }
        else
        {
            return 2;
        }
    }
    else
    {
        outputFile << pre << " " << pre << "\n";
        ch = nxt;
        return 2;
```


教务处制

```
        }

        return 2;
    }

    int handleInvalidToken()
    {
        outputFile << "[Error] Invalid token! line:" << line << " col:" << col << "\n";
        std::cout<<"[Error] Invalid token! line:" << line << " col:" << col << "\n";
        return 1;
    }

    void handleID(std::string temp)
    {
         std::string id = temp;

        for(auto &i:id)
        {
                if (i >= 'A' && i <= 'Z')
                i += 'a' - 'A';

        }

        for (auto i : keywords)
        {
            if (id == i)
            {
                outputFile << i << " " << i << "\n";
                return;
            }
        }

          outputFile << "ID " << temp << "\n";
    }
};

int main()
{

    Lexer lexer("AAA.txt", "AAA1.out");
    if(lexer.run())
        std::cout << "Error!\n";
    else
        std::cout << "Completed!\n";
```

```
    return 0;
}
```


教务处制