电子科技大学

计算机专业类课程

实验报告

课程名称：编译原理

学　　院：计算机科学与工程学院

专　　业：计算机科学与技术

学生姓名：李宇潇

学　　号：2021080907032

指导教师：陈昆

日　　期：　2024 年　6 月　6 日

电 子 科 技 大 学

实　验　报　告

实验一

一、实验名称：词法分析

二、实验学时：4

三、实验内容和目的：

求 n！的词法分析，要求根据测试程序生成词法分析结果，并且得出二元式文件输出和错误文件输出，要求词法分析器需要报三种错误：1 非法字符；2 冒号不匹配；3 标识符长度溢出

四、实验原理：

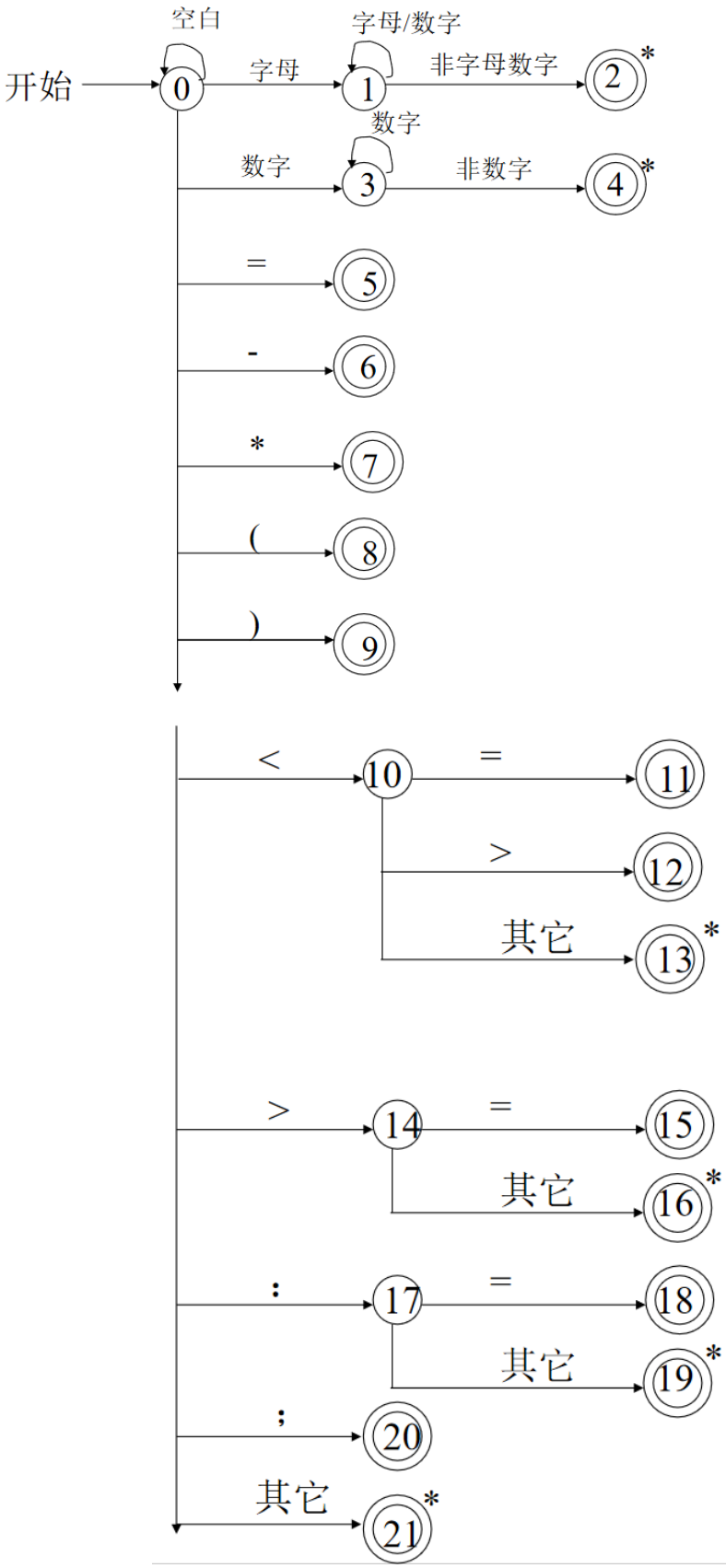
词法分析（Lexical Analysis）是编译过程的第一步，旨在将源代码转换为一系列有意义的词法单元（Tokens）。词法分析器扫描源代码的字符流，识别出构成程序基本元素的词法单元，如标识符、关键字、运算符和标点符号。

其核心原理包括：

1. **正则表达式匹配**：使用正则表达式定义各种词法单元的模式，通过模式匹配识别和提取这些单元。
2. **有限状态机（FSM）**：将正则表达式转换为有限状态自动机，通过状态转移识别输入字符序列的结构。
3. **消除空白和注释**：忽略无意义的空白字符和注释，以确保只分析实际的源代码内容。

词法分析的目标是简化后续语法分析过程，使其能够更高效地理解代码结构和语义。

在该实验中我所构建的状态图为教材提供的状态转换图，如下所示：



其中：双圆圈表示匹配中止，带有*号表示需要回退一字符。

五、实验器材（设备、元器件）

笔记本电脑

六、实验步骤：

1、分析用 C 语言实现词法分析的过程：

完整的词法分析器至少包括以下几个过程
1）文件打开和读取（.pas 文件）
2）使用有限状态机判断当前读取的符号的类别，需要特殊关注行结束符和文件结束符。
3）将二元式输出到结果文件（.dyd 文件）

2、用 c 语言实现各个过程

1）使用 fopen 函数打开对应文件

```
 source = fopen("E:\\C++\\Compile\\source.pas", "r");
dyd = fopen("E:\\C++\\Compile\\output.dyd", "w");
err = fopen("E:\\C++\\Compile\\output.err", "w");
```

并在无法打开文件时退出程序

```
if (!source || !dyd || !err)
{
    printf("Error opening files.\n");
    return;
}
```

2）设计 get_token 函数依次读取文件，使用状态转换机判断符号所属类别

使用 char ch = fgetc()可以读取文件中的一个字符，包括换行字符，每次读取之后对 ch 进行判断，有如下几种情况：

a. isspace(ch) && ch != ‘\n’：此时为空白符号，直接跳过，读取下一个字符。

```
while (isspace(ch) && ch != '\n')
{
    ch = fgetc(source);
}
```

b. ch == ‘\n’：此时为换行符，则插入一个行结束符。

```
if (ch == '\n')
{
    line_number++;
    Token newline_token = {24, "EOLN", line_number};
    return newline_token;
}
```

c. ch == EOF：此时为文件结尾，则插入一个文件结束符。

```
if (ch == EOF){
    token.type = 25;
```

```
        strcpy(token.value, "EOF");
        token.line = line_number;
        return token;
    }
```

d. isalpha(ch)：此时为一个字母，后续使用 buffer 存放后续字符，直到后续字符不会字母或者数字，最后再将 buffer 中存放的符号与几个保留字比较，如果未匹配到保留字则字符种类为标识符，如果有匹配的保留字，则类别为保留字所属的类别，但如果超过了 buffer 最大长度则出现**标识符长度溢出错误**。

```
    if (isalpha(ch)){
        int length = 0;
        char buffer[MAX_ID_LENGTH + 1];
        while (isalnum(ch))
        {
            if (length < MAX_ID_LENGTH)
            {
                buffer[length++] = ch;
            }
            else
            {
                report_error("Identifier length overflow", line_number);
                while (isalnum(ch))
                    ch = fgetc(source);
                break;
            }
            ch = fgetc(source);
        }
        ungetc(ch, source);
        buffer[length] = '\0';
        strcpy(token.value, buffer);
        token.line = line_number;
        // Check for reserved words
        token.type = 10;
        if (strcmp(token.value, "begin") == 0)
            token.type = 1;
        else if (strcmp(token.value, "end") == 0)
            token.type = 2;
        else if (strcmp(token.value, "integer") == 0)
            token.type = 3;
        else if (strcmp(token.value, "function") == 0)
            token.type = 7;
        else if (strcmp(token.value, "read") == 0)
            token.type = 8;
        else if (strcmp(token.value, "write") == 0)
            token.type = 9;
```

```
        else if (strcmp(token.value, "if") == 0)
            token.type = 4;
        else if (strcmp(token.value, "then") == 0)
            token.type = 5;
        else if (strcmp(token.value, "else") == 0)
            token.type = 6;
    }
```

e. isdigit(ch)：此时为一个数字，使用 buffer 存放后续数字直到下一个字符不会数字，设置类别为数字。

```
    if (isdigit(ch))
    {
        int length = 0;
        char buffer[MAX_ID_LENGTH + 1];
        while (isdigit(ch))
        {
            buffer[length++] = ch;
            ch = fgetc(source);
        }
        ungetc(ch, source);
        buffer[length] = '\0';

        token.type = 11;
        strcpy(token.value, buffer);
        token.line = line_number;
    }
```

f. ch == ‘:’：考虑匹配赋值符号，如果无法匹配为:=则出现**冒号不匹配错误。**

```
    else if (ch == ':')
    {
        ch = fgetc(source);
        if (ch == '=')
        {
            token.type = 20;
            strcpy(token.value, ":=");
        }
        else
        {
            ungetc(ch, source);
            report_error("Colon not matched", line_number);
            token.type = 0;
            strcpy(token.value, ":");
        }
        token.line = line_number;
    }
```

g. ch == ‘<’：考虑匹配<>, <=, <符号。

```
else if (ch == '<')
{
    ch = fgetc(source);
    if (ch == '>')
    {
        token.type = 13;
        strcpy(token.value, "<>");
    }
    else if (ch == '=')
    {
        token.type = 14;
        strcpy(token.value, "<=");
    }
    else
    {
        ungetc(ch, source);
        token.type = 15;
        strcpy(token.value, "<");
    }
    token.line = line_number;
}
```

h. ch == ‘>’：考虑匹配>=, >符号。

```
else if (ch == '>')
{
    ch = fgetc(source);
    if (ch == '=')
    {
        token.type = 16;
        strcpy(token.value, ">=");
    }
    else
    {
        ungetc(ch, source);
        token.type = 17;
        strcpy(token.value, ">");
    }
    token.line = line_number;
```

i. ch == ‘=’||‘-’||‘*’||‘（’||‘）’||‘；’：匹配对应的符号。

```
else if (ch == '=')
{
    token.type = 12;
```

```
        strcpy(token.value, "=");
        token.line = line_number;
    }
    else if (ch == '-'){
        token.type = 18;
        strcpy(token.value, "-");
        token.line = line_number;
    }
    else if (ch == '*')
    {
        token.type = 19;
        strcpy(token.value, "*");
        token.line = line_number;
    }
    else if (ch == '(')
    {
        token.type = 21;
        strcpy(token.value, "(");
        token.line = line_number;
    }
    else if (ch == ')')
    {
        token.type = 22;
        strcpy(token.value, ")");
        token.line = line_number;
    }
    else if (ch == ';')
    {
        token.type = 23;
        strcpy(token.value, ";");
        token.line = line_number;
    }
```

如果以上都不符合，那么出现**非法符号错误。**插入行结束符是为了在后续语法分析过程中快速识别当前字符所在行数，直观地展示在报错文件中。

```
    else
    {
        char illegal_char[2];
        illegal_char[0] = ch;
        illegal_char[1] = '\0';
        report_error("Illegal character", line_number);
        token.type = 0;
        strcpy(token.value, illegal_char);
        token.line = line_number;
    }
```

3）设计函数不断调用 get_token 函数，判断返回符号种类，并写入输出文件

```
    while (1)
```

```
    {
        token = get_token();
        if (token.type == 25)
        {
            fprintf(dyd, "     ... EOF 25\n");//文件结束符
            break;
        }
        else if (token.type == 24)
        {
            fprintf(dyd, "     ... EOLN 24\n");//行结束符号
            current_line++;
        }
        else if (token.type != 0)
        {
            fprintf(dyd, "%s %d\n", token.value, token.type);
        }
    }
```

4）设计报错信息

将报错信息传给 report_error 函数，在函数内部将信息写入到.err 文件中

```
void report_error(const char *message, int line)
{
    fprintf(err, "***LINE:%d   %s\n", line, message);
}
```

七、实验数据及结果分析：

实验要求词法分析器需要报三种错误：1 非法字符；2 冒号不匹配；3 标识符长度溢出，下面设计不同的测试程序，使其：无词法错误，有非法字符，有冒号不匹配，有标识符长度溢出几种错误，观察分析结果，下图中**左侧为测试程序，中间为 dyd 文件，右侧为 err 文件**。

1、无词法错误

```
begin
  integer m;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  read(m);
  k:=F(m);
  write(k);
end;
```

```
begin 1
    ... EOLN 24
integer 3
m 10
; 23
    ... EOLN 24
integer 3
function 7
F 10
( 21
n 10
) 22
; 23
    ... EOLN 24
begin 1
    ... EOLN 24
integer 3
n 10
```

```
1
```

2、有冒号不匹配

```
1  begin
2      integer m;
3      integer function F(n);
4        begin
5          integer n;
6          if n<=0 then F:1;
7          else F:=n*F(n-1);
8        end;
9      read(m);
10     k:=F(m);
11     write(k);
12   end;
```

```
n 10
) 22
; 23
    ... EOLN 24
begin 1
    ... EOLN 24
integer 3
n 10
```

```
1  ***LINE:6  Colon not matched
2
```

3、有标识符长度溢出

```
1  begin
2    integer abcdefghijklmnopqrstuvwxyz;
3    integer function F(n);
4      begin
5        integer n;
6        if n<=0 then F:=1;
7        else F:=n*F(n-1);
8      end;
9    read(m);
10   k:=F(m);
11   write(k);
12 end;
```

```
1   begin 1
2       ... EOLN 24
3   integer 3
4   abcdefghijklmnop 10
5   ; 23
6       ... EOLN 24
7   integer 3
8   function 7
9   F 10
10  ( 21
11  n 10
12  ) 22
13  ; 23
14      ... EOLN 24
15  begin 1
16      ... EOLN 24
17  integer 3
18  n 10
```

```
1   ***LINE:2  Identifier length overflow
2
```

4、有非法符号

```
1   begin
2     integer m;&
3     integer function F(n);
4       begin
5         integer n;
6         if n<=0 then F:=1;
7         else F:=n*F(n-1);
8       end;
9     read(m);
10    k:=F(m);
11    write(k);
12  end;
```

```
begin 1
   ... EOLN 24
integer 3
m 10
; 23
   ... EOLN 24
integer 3
function 7
F 10
( 21
n 10
) 22
; 23
   ... EOLN 24
begin 1
   ... EOLN 24
integer 3
n 10
```

```
1   ***LINE:2  Illegal character
2
```

从上面的几个测试程序和结果可以看出，该词法分析器可以在出现冒号不匹配，非法字符以及标识符溢出的情况下正确识别出词法错误，并在词法无错误时正确运行，表现了该程序和该实验的成功。

八、实验结论、心得体会和改进建议：

最开始考虑词法分析器的设计时感觉逻辑比较简单，但是实际编写代码的时候发现仍然存在很多问题，比如怎么将源代码分成一个一个 token 放入所设计的数据结构，怎么读取换行符号等等。同时有限状态机的设计也并不像想像中那么简单，在读取每一个字符的时候都要仔细考虑是否正确，是否多读取了字符或者是否需要使用 ungetc 将字符放回文件中，又比如如何记录当前字符所在的行等等。并最终还是正确完成了词法分析器，通过这个实验我对词法分析过程和有限状态机的理解都更加深入，也对编译的完整过程有了更多兴趣，推动着我完成后续的语法分析等实验。

建议老师更加明确地表明每个实验的具体要求，比如在该词法分析实验中，在 ppt 的各阶段的输入输出部分只写了词法分析输入为.pas 代码，输出为.dyd 二元式，但是根据后面的要求来看，该实验仍然需要输出.err 错误文件来查看错误位置。并且该 err 文件为后续每个阶段共用的文件，在每个阶段都需要查看该文件是否为空，只有为空时才能继续执行持续分析阶段。另外在实验初期我存在**一个疑惑**，就是老师要求了在每行后面加行结束符，这就导致我以为二元式的形式是按照源代码中符号的位置，只有源程序换行时二元式才换行并加入行结束符。但是后续我查阅了其他资料之后发现二元式每一行是一个二元式，换行符单独占一行即可，所以希望老师可以在后续的实验中稍作解释。**非常感谢老师的阅读，祝老师工作顺利，身体健康！**

电 子 科 技 大 学

实　验　报　告

实验二

一、实验名称：语法分析

二、实验学时：4

三、实验内容和目的：

求 n！的语法分析，要求根据没有词法错误的词法分析结果进行语法分析，并且要求先消除文法中的左递归，使用递归下降法来进行语法分析，在分析过程中得到语法分析的结果，变量名表，过程名表以及报错信息。

四、实验原理：

下面展示消除了文法左递归之后的文法，由于在实验过程中我认为分号是语句结束的标志，每条语句都需要加上分号，所以我对文法进行了一点修改，要求在说明语句和执行语句后面都匹配分号，只有最后一个 end 不匹配分号。老师没有给出函数调用的进一步文法，我自己补为其推出<变量>(<算术表达式>)，消除左递归之后的文法如下：

<程序> → <分程序>
<分程序> → begin <说明语句表><执行语句表> end
<说明语句表> → <说明语句> <说明语句表'>
<说明语句表'> → <说明语句> <说明语句表'> | ε
<说明语句> → <变量说明>; | <函数说明>;
<变量说明> → integer <变量>
<变量> → <标识符>
<标识符> → <字母> | <标识符><字母> | <标识符><数字>
<字母> → a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<数字> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<函数说明> → integer function <标识符>(<参数>); <函数体>
<参数> → <变量>
<函数体> → begin <说明语句表><执行语句表> end
<执行语句表> → <执行语句> <执行语句表'>
<执行语句表'> → <执行语句> <执行语句表'> | ε
<执行语句> → <读语句> | <写语句> | <赋值语句> | <条件语句>
<读语句> → read(<变量>);
<写语句> → write(<变量>);
<赋值语句> → <变量> := <算术表达式>;
<算术表达式> → <项> <算术表达式'>

<算术表达式'> → - <项> <算术表达式'> | ε
<项> → <因子> <项'>
<项'> → * <因子> <项'> | ε
<因子> → <变量> | <常数> | <函数调用>
<函数调用> → <变量>(<算术表达式>)
<常数> → <无符号整数>
<无符号整数> → <数字> | <无符号整数><数字>
<条件语句> → if <条件表达式> then <执行语句> else <执行语句>
<条件表达式> → <算术表达式> <关系运算符> <算术表达式>
<关系运算符> → < | <= | > | >= | = | <>
根据上述消除左递归之后的文法即可编写代码完成语法分析过程。

五、实验器材（设备、元器件）

笔记本电脑

六、实验步骤：

1.分析语法分析过程所需要设计的函数和数据结构
1) add_variable: 向变量名表中增加变量
2) is_variable_exists: 查找变量名表中是否存在层次和类型都相同的变量，主要用于检查变量是否声明过
3) print_tables: 将变量名表和过程名表写入对应文件中
4) get_token: 读取 dyd 二元式文件，并将字符全部存放在 token arrays 中，方便后续使用
5) getNextToken, peekNextToken: 用于读取下一个 token，两者区别在于是否移动 token 索引编号
6) syntaxError: 用于输出报错信息到报错文件，同时决定该错误是否严重，如果不严重则可以继续执行后续的。在该实验中由于没有找到很好的办法来处理符号不匹配或者缺失所导致符号错位的问题，所以只有当是变量未定义或重复定义这种比较不严重的错误可以继续分析，其他不匹配或者缺失的错误都记录报错信息然后退出程序
7) match: 用来匹配下一个 token 和预期 token，如果无法匹配则记录报错信息。
8) 递归下降分析法的解析函数，具体步骤不在此中展示，详细见文末代码。
实验中需要使用的数据结构包括：变量名，过程名以及 token。具体数据结构定义如下：

```
#define MAX_ID_LENGTH 16
typedef struct
{
    char vname[MAX_ID_LENGTH + 1];
    char vproc[MAX_ID_LENGTH + 1];
    int vkind; // 0 for variable, 1 for parameter
    char vtype[10];
    int vlev;
    int vadr;
```

```
} VarEntry;
typedef struct
{
    char pname[MAX_ID_LENGTH + 1];
    char ptype[10];
    int plev;
    int fadr;
    int ladr;
} FuncEntry;
typedef struct
{
    char value[MAX_ID_LENGTH + 1];
    int type;
    int line;
} Token;
```

2、编写各个函数中所需代码

由于本文法内容较多，不在此全部阐述，只介绍几个比较关键的思想。

1）pascal 语言文法整体遵循 begin 开始，说明语句，执行语句，end 结尾的思路，所以说明语句之后要么接说明语句要么接执行语句，而执行语句之后只能跟执行语句或者结束标志。

2）对于空行需要特殊处理，由于空行可能出现了很多地方，而且一般不会影响程序的正确性，所以我编写了一个 new_line_test 函数，在匹配过程中首先调用这个函数来测试下一个 token 是否为换行符，如果是则读取该换行符并且当前行数加 1 并且递增 token 索引数，如果不是则正常匹配。

3）在说明语句中，包含变量说明和函数说明，并且两者都以 integer 开头，所以这里需要预读两个 token 来判断匹配符号，在 peekNextToken 函数中有一偏移参数，可以传入需要预读的位置，比如 0 则返回下一个 token，1 则返回下两个 token。

4）我以自己的理解设计了变量名表和过程名表，当遇到变量和函数声明时，向变量和过程名表中加入该函数或变量，同时根据 begin 的嵌套次数来决定当前的 level，根据当前变量个数设置 fadr 数，每次增加函数名的时候都在过程名表中的所属过程递增 ladr 数。同时在增加函数过程时，由于后续在函数体中函数名也可作为函数的返回值，所以需要同时在变量名表和过程名表中增加函数名这个符号，同时由于 pascal 语言允许外层调用内层函数，所以需要在各个更低的 level 都增加该符号的记录，具体结果见后面结果验证部分。

5）在变量声明过程中检查变量是否已经被声明过，若是则报变量重复定义错。而在变量解释中，检查变量是否已经被声明过，若不是则报变量未定义错。这两类错误属于不严重文法错误，遇到之后语法分析过程可以继续进行。

6）在解析执行语句函数中，预读取一个 token 来判断属于哪类执行语句，如果四类执行语句（condition,read,write,assign）都无法匹配则报执行语句匹配错误。

7）在二元式中，存储的信息为字符和种别码，但是在语法分析报错中为了方便直观了解报错原因，不能直接展示种别码，所以需要使用一个映射将种别码翻译成对应字符再输出到报错文件中。

七、实验数据及结果分析：

下面针对几种不同的语法错误对该程序进行测验，并展示运行结果，在下图中从左到右，从上到下依次是原程序，报错信息，变量名表，过程名表：

1、无错误的程序

```
begin
  integer k;
  integer m;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  read(m);
  k:=F(m);
  write(k);
end
```

```
1
```

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: m, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 1
Name: F, Proc: F, Kind: 0, Type: default, Level: 1, Addr: 2
Name: F, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 3
Name: n, Proc: F, Kind: 1, Type: default, Level: 2, Addr: 4
Name: n, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 5

```

```
Name: F, Type: default, Level: 2, First Addr: 2, Last Addr: 5

```

2、有不严重的错误（变量未定义，重复定义）

```
begin
  integer k;
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  read(m);
  k:=F(m);
  write(k);
end
```

```
***3: Variable k is already defined
***10: Variable m is not defined
***11: Variable m is not defined

```

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: F, Proc: F, Kind: 0, Type: default, Level: 1, Addr: 1
Name: F, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 2
Name: n, Proc: F, Kind: 1, Type: default, Level: 2, Addr: 3
Name: n, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 4

```

```
Name: F, Type: default, Level: 2, First Addr: 1, Last Addr: 4

```

3、有严重错误（匹配错误，符号缺失，执行语句部分出现说明语句）

1）

```
begin
  integer k;
  integer funcion F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  read(m);
  k:=F(m);
  write(k);
end
```

```
***3: Expected token ";" but got "F"

```

这里由于将 funtion 当作一个变量，所以应该匹配分号，但是实际出现了 F，所以报错无误。

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: funcion, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 1

```

```
1
```

2）

```
begin
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n;-1);
    end;
  read(m);
  k:=F(m);
  write(k);
end
```

```
***7: Expected token ")" but got ";"

```

这里为函数调用的报错，在函数调用中预期匹配一个表达式，而 n 已经匹配了表达式，下一个应该匹配右括号，但是却遇到分号，所以报符号不匹配错误。

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: F, Proc: F, Kind: 0, Type: default, Level: 1, Addr: 1
Name: F, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 2
Name: n, Proc: F, Kind: 1, Type: default, Level: 2, Addr: 3
Name: n, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 4

```

```
Name: F, Type: default, Level: 2, First Addr: 1, Last Addr: 4

```

3）

```
begin
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  read(m);
  integer m;
  k:=F(m);
  write(k);
end
```

```
***9: Variable m is not defined
***10: Expected token "end" but got "integer"

```

这里由于在执行语句之后只能是执行语句或者 end 符号，而 integer 无法匹配到任何一个执行语句（condition,read,write,assign）所以只能尝试匹配 end，所以报错无误。

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: F, Proc: F, Kind: 0, Type: default, Level: 1, Addr: 1
Name: F, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 2
Name: n, Proc: F, Kind: 1, Type: default, Level: 2, Addr: 3
Name: n, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 4

```

```
Name: F, Type: default, Level: 2, First Addr: 1, Last Addr: 4

```

4）

```
begin
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1;
      else F:=n*F(n-1);
    end;
  readm);
  k:=F(m);
  write(k);
end
```

```
***9: Variable readm is not defined
***9: Expected token ":=" but got ")"
```

这里将 readm 当作一个变量，所以匹配到的执行语句为 assign，但是没有:=符号，所以报匹配错误。

```
Name: k, Proc: main, Kind: 0, Type: default, Level: 1, Addr: 0
Name: F, Proc: F, Kind: 0, Type: default, Level: 1, Addr: 1
Name: F, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 2
Name: n, Proc: F, Kind: 1, Type: default, Level: 2, Addr: 3
Name: n, Proc: F, Kind: 0, Type: default, Level: 2, Addr: 4
```

```
Name: F, Type: default, Level: 2, First Addr: 1, Last Addr: 4
```

通过以上四个测试程序可以看出，该方法分析程序可以正确处理绝大部分的文法错误情况，并且报出正确的错误，说明该程序和本实验的成功！

八、实验结论、心得体会和改进建议：

在完成该实验中遇到了非常多的困难，受限于本人对变量名表和过程名表的生疏，即便最终也没有非常完美地完成整个实验，但是我已经尽我最大的努力来完成这个实验并且学习到了非常多的东西。首先我体会到完成该实验必须要求我们对 pascal 语言的文法非常熟悉，当在源程序中进行任意修改的时候都可以反应过来出现的错误是什么，当我们对文法足够熟悉之后才能成功地编写代码，不然在调度代码的时候自己都不知道应该跳转到什么函数或者应该报什么错，就更加没有办法编写代码了。另外在消除文法左递归的时候发现了一些难以理解的问题，比如为什么一会要分号，一会不要分号，在和老师交流之后才明白在 pascal 语言中是以没有分号作为执行语句表结束的标志，由于一开始实验没有考虑到这个问题，也对 pascal 的规则不太了解，所以简单地修改了文法，让每个执行语句都需要匹配分号，但是还是要求最后的 end 没有分号，这样就相当于让 end 作为执行语句结束地标志。同时我根据自己对变量名表和过程名表的理解对这两者的代码做了相应的处理，尽可能地满足了实验要求。

建议：在我做实验的时候有一些**疑惑**，并且我与同学们交流之后发现大家都不太了解，所以希望老师之后可以进一步阐述一下，就是由于老师给的测试程序中是存在错误的，所以我们不太清楚对于 pascal 而言正确的文法是什么，比如到底什么地方需要分号，什么地方不需要，这就导致我们阅读老师给出的文法的时候有一些疑惑，甚至怀疑老师给的文法到底是否正确从而修改成自己认为正确的文法。所以老师如果能稍微解释一下 pascal 的测试程序中存在哪些错误，或者给出一个完全正确的 pascal 程序，比如解释一下在严格的 pascal 语法中中分号缺失代表着执行语句表的结束，我们将会对整个语法分析过程有更加深入的理解并且更加快速明确地完成实验。同时我们同学交流的时候发现大家对变量名表和过程名表都不太熟悉，希望老师可以对这些难度比较大的部分做一些解释或者提示。**非常感谢老师的阅读，祝老师工作顺利，身体健康！**

实验代码：

1、词法分析

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

#define MAX_ID_LENGTH 16

typedef struct
{
    int type;
    char value[MAX_ID_LENGTH + 1];
    int line;
} Token;

FILE *source, *dyd, *err;
int line_number = 1;

void report_error(const char *message, int line)
{
    fprintf(err, "***LINE:%d   %s\n", line, message);
}



Token get_token()
{
    Token token;
    int ch = fgetc(source);

    while (isspace(ch) && ch != '\n')
    {
        ch = fgetc(source);
    }

    if (ch == '\n')
    {
        line_number++;
        Token newline_token = {24, "EOLN", line_number};
        return newline_token;
    }
```

```
    if (ch == EOF)
    {
        token.type = 25;
        strcpy(token.value, "EOF");
        token.line = line_number;
        return token;
    }

    if (isalpha(ch))
    {
        int length = 0;
        char buffer[MAX_ID_LENGTH + 1];
        while (isalnum(ch))
        {
            if (length < MAX_ID_LENGTH)
            {
                buffer[length++] = ch;
            }
            else
            {
                report_error("Identifier length overflow", line_number);
                while (isalnum(ch))
                    ch = fgetc(source);
                break;
            }
            ch = fgetc(source);
        }
        ungetc(ch, source);
        buffer[length] = '\0';
        strcpy(token.value, buffer);
        token.line = line_number;
        // Check for reserved words
        token.type = 10;
        if (strcmp(token.value, "begin") == 0)
            token.type = 1;
        else if (strcmp(token.value, "end") == 0)
            token.type = 2;
        else if (strcmp(token.value, "integer") == 0)
            token.type = 3;
        else if (strcmp(token.value, "function") == 0)
            token.type = 7;
        else if (strcmp(token.value, "read") == 0)
            token.type = 8;
        else if (strcmp(token.value, "write") == 0)
```

```
            token.type = 9;
        else if (strcmp(token.value, "if") == 0)
            token.type = 4;
        else if (strcmp(token.value, "then") == 0)
            token.type = 5;
        else if (strcmp(token.value, "else") == 0)
            token.type = 6;
    }
    else if (isdigit(ch))
    {
        int length = 0;
        char buffer[MAX_ID_LENGTH + 1];
        while (isdigit(ch))
        {
            buffer[length++] = ch;
            ch = fgetc(source);
        }
        ungetc(ch, source);
        buffer[length] = '\0';

        token.type = 11;
        strcpy(token.value, buffer);
        token.line = line_number;
    }
    else if (ch == ':')
    {
        ch = fgetc(source);
        if (ch == '=')
        {
            token.type = 20;
            strcpy(token.value, ":=");
        }
        else
        {
            ungetc(ch, source);
            report_error("Colon not matched", line_number);
            token.type = 0;
            strcpy(token.value, ":");
        }
        token.line = line_number;
    }
    else if (ch == '<')
    {
        ch = fgetc(source);
        if (ch == '>')
        {
            token.type = 13;
```

```
            strcpy(token.value, "<>");
        }
        else if (ch == '=')
        {
            token.type = 14;
            strcpy(token.value, "<=");
        }
        else
        {
            ungetc(ch, source);
            token.type = 15;
            strcpy(token.value, "<");
        }
        token.line = line_number;
    }
    else if (ch == '>')
    {
        ch = fgetc(source);
        if (ch == '=')
        {
            token.type = 16;
            strcpy(token.value, ">=");
        }
        else
        {
            ungetc(ch, source);
            token.type = 17;
            strcpy(token.value, ">");
        }
        token.line = line_number;
    }
    else if (ch == '=')
    {
        token.type = 12;
        strcpy(token.value, "=");
        token.line = line_number;
    }
    else if (ch == '-'){
        token.type = 18;
        strcpy(token.value, "-");
        token.line = line_number;
    }
    else if (ch == '*')
    {
```

```
        token.type = 19;
        strcpy(token.value, "*");
        token.line = line_number;
    }
    else if (ch == '(')
    {
        token.type = 21;
        strcpy(token.value, "(");
        token.line = line_number;
    }
    else if (ch == ')')
    {
        token.type = 22;
        strcpy(token.value, ")");
        token.line = line_number;
    }
    else if (ch == ';')
    {
        token.type = 23;
        strcpy(token.value, ";");
        token.line = line_number;
    }
    else
    {
        char illegal_char[2];
        illegal_char[0] = ch;
        illegal_char[1] = '\0';
        report_error("Illegal character", line_number);
        token.type = 0;
        strcpy(token.value, illegal_char);
        token.line = line_number;
    }

    return token;
}

void lexical_analysis()
{
    Token token;
    source = fopen("E:\\C++\\Compile\\source.pas", "r");
    dyd = fopen("E:\\C++\\Compile\\output.dyd", "w");
    err = fopen("E:\\C++\\Compile\\output.err", "w");

    if (!source || !dyd || !err)
    {
        printf("Error opening files.\n");
        return;
```

```
    }

    int current_line = 1;
    while (1)
    {
        token = get_token();
        if (token.type == 25)
        {
            fprintf(dyd, "EOF 25\n");
            break;
        }
        else if (token.type == 24)
        {
            fprintf(dyd, "EOLN 24\n");
            current_line++;
        }
        else if (token.type != 0)
        {
            fprintf(dyd, "%s %d\n", token.value, token.type);
        }
    }

    fclose(source);
    fclose(dyd);
    fclose(err);
}

int main()
{
    lexical_analysis();
    return 0;
}
```

2、语法分析

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_ID_LENGTH 16
typedef struct
{
    char vname[MAX_ID_LENGTH + 1];
    char vproc[MAX_ID_LENGTH + 1];
    int vkind; // 0 for variable, 1 for parameter
    char vtype[10];
    int vlev;
    int vadr;
} VarEntry;
typedef struct
{
    char pname[MAX_ID_LENGTH + 1];
    char ptype[10];
    int plev;
    int fadr;
    int ladr;
} FuncEntry;
typedef struct
{
    char value[MAX_ID_LENGTH + 1];
    int type;
    int line;
} Token;

FILE *file, *err, *output, *pro, *var;
VarEntry varTable[100];
FuncEntry funcTable[100];
Token tokens[1000];

int varCount = 1;
int funcCount = 1;
int level = 0;
int tokenCount = 0;
int tokenLine = 1;
int currentTokenIndex = 0;
char typetab[][20] = {{""},{"begin"}, {"end"}, {"integer"}, {"if"}, {"then"}, {"else"}, {"function"}, {"read"},
{"write"}, {"(identifier)"}, {"(number)"}, {"="}, {"<>"}, {"<="}, {"<"}, {">="}, {">"}, {"-"}, {"*"}, {":="},
{"("}, {")"}, {";"}, {"EOLN"}, {"EOF"}};
```

```
void add_variable(const char *name, const char *proc, int kind, const char *type = 0, int lev = 0)
{

    strcpy(varTable[varCount].vname, name);
    strcpy(varTable[varCount].vproc, proc);
    varTable[varCount].vkind = kind;
    if (type == 0)
    {
        strcpy(varTable[varCount].vtype, "default");
    }
    else
        strcpy(varTable[varCount].vtype, type);
    varTable[varCount].vlev = lev;
    varTable[varCount].vadr = varCount - 1;
    if (lev > 1)
    {
        funcTable[lev - 1].ladr = varCount - 1;
    }
    varCount++;
}
bool is_variable_exists(const char *name, int level, int kind)
{
    for (int i = 0; i < varCount; i++)
    {
        if ((strcmp(varTable[i].vname, name) == 0) && (varTable[i].vlev == level) && (varTable[i].vkind ==
kind))
        {
            return true;
        }
    }
    return false;
}
void close_file()
{
    fclose(file);
    fclose(err);
    fclose(output);
    fclose(pro);
    fclose(var);
}
void add_function(const char *name, const char *type, int lev, int fadr, int ladr)
{
    strcpy(funcTable[funcCount].pname, name);
    strcpy(funcTable[funcCount].ptype, type);
```

```
    funcTable[funcCount].plev = lev;
    funcTable[funcCount].fadr = fadr;
    funcTable[funcCount].ladr = ladr;
    funcCount++;
}
void print_tables();
Token getNextToken()
{
    return tokens[currentTokenIndex++];
}

Token peekNextToken(int i)
{
    return tokens[currentTokenIndex + i];
}

void syntaxError(const char *message, int type)
// type 0 for warning, -1 for error
{
    fprintf(err, "%s\n", message);
    if (type == -1)
    {
        print_tables();
        close_file();
        exit(1);
    }
}

void match(int expectedType)
{
    Token token = getNextToken();
    while (token.type == 24)
    {
        fprintf(output, "%s %d\n", token.value, token.type);
        token = getNextToken();
    }
    if (token.type != expectedType)
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Expected token \"%s\" but got \"%s\"", token.line, typetab[expectedType],
token.value);
        syntaxError(errorMsg, -1);
    }
    else
    {
        fprintf(output, "%s %d\n", token.value, token.type);
    }
```

```
}

void parseProgram();
void parseSubProgram();
void parseDeclarationList();
void parseDeclarationListPrime();
void parseDeclaration();
void parseVariableDeclaration();
void parseVariable(int type);
void parseFunctionDeclaration();
void parseIdentifier();
void parseIdentifierPrime();
void parseStatementList();
void parseStatementListPrime();
void parseStatement();
void parseExpression();
void parseExpressionPrime();
void parseTerm();
void parseTermPrime();
void parseFactor();
void parseConditionExpression();
void parseRelationOperator();

void new_line_test()
{
    Token token = peekNextToken(0);
    if (token.type == 24)
    {
        fprintf(output, "%s %d\n", token.value, token.type);
        getNextToken();
    }
}
void parseProgram()
{
    new_line_test();
    parseSubProgram();
    match(25);
}
void parseSubProgram()
{
    match(1);
    level++; // level++ when match begin
    parseDeclarationList();
    parseStatementList();
```

```
    match(2);
    level--; // level-- when match end
}
void parseDeclarationList()
{
    new_line_test();
    parseDeclaration();
    parseDeclarationListPrime();
}
void parseDeclarationListPrime()
{
    new_line_test();
    Token token = peekNextToken(0);
    if (token.type == 3) // variable declaration and function declaration both start with 'integer'
    {
        parseDeclaration();
        parseDeclarationListPrime();
    }
}
void parseDeclaration()
{
    new_line_test();
    Token token1 = peekNextToken(0);
    Token token2 = peekNextToken(1);
    if (token1.type == 3 && token2.type == 10) // variable declaration
    {
        parseVariableDeclaration();
        match(23);
    }
    else if (token1.type == 3 && token2.type == 7) // function declaration
    {
        parseFunctionDeclaration();
        match(23);
    }
    else
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Invalid declaration", token1.line);
        syntaxError(errorMsg, -1);
    }
}
void get_proc_name(int level, char *name)
{
    new_line_test();
    int index = level - 1;
    if (index == 0)
    {
```

```
        strcpy(name, "main");
    }
    else
    {
        strcpy(name, funcTable[index].pname);
    }
}
void parseVariableDeclaration()
{
    new_line_test();
    match(3);
    Token token = getNextToken();
    if (token.type != 10)
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Expected identifier in variable declaration", token.line);
        syntaxError(errorMsg, -1);
    }
    else if (is_variable_exists(token.value, level, 0))
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Variable %s is already defined", token.line, token.value);
        syntaxError(errorMsg, 0);
    }
    else
    {
        fprintf(output, "%s %d\n", token.value, token.type);
        char proc_name[17];
        get_proc_name(level, proc_name);
        add_variable(token.value, proc_name, 0, 0, level);
    }
    // add variable to variable table

}
void parseVariable(int type)
{
    new_line_test();
    Token token = getNextToken();
    if (token.type != 10)
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Expected identifier", token.line);
        syntaxError(errorMsg, -1);
    }
```

```
    if (type == 0) // check if the variable exists
    {
        if (!is_variable_exists(token.value, level, 0))
        {
            char errorMsg[128];
            sprintf(errorMsg, "***%d: Variable %s is not defined", token.line, token.value);
            fprintf(output, "%s %d\n", token.value, token.type);
            syntaxError(errorMsg, 0);
        }
        else
        {
            fprintf(output, "%s %d\n", token.value, token.type);
        }
    }
    else if (type == 1) // formal parameters
    {
        char proc_name[17];
        get_proc_name(level + 1, proc_name);
        add_variable(token.value, proc_name, 1, 0, level + 1);
        fprintf(output, "%s %d\n", token.value, token.type);
    }
    else if (type == 2) // function
    {
        add_function(token.value, "default", level + 1, varCount - 1, varCount - 1);
        char proc_name[17];
        get_proc_name(level + 1, proc_name); // function name is the return value as well
        for (int i = 1; i <= level + 1; i++)
        { // add the function name to the variable table with lower level
            add_variable(token.value, proc_name, 0, 0, i);
        }

        fprintf(output, "%s %d\n", token.value, token.type);
    }
}
void parseFunctionDeclaration()
{
    new_line_test();
    match(3);
    match(7);
    parseVariable(2); // Function variable

    match(21); // LPAREN
    parseVariable(1);
    match(22); // RPAREN
    match(23); // SEMICOLON
    parseSubProgram();
}
```

```
void parseStatementList()
{
    new_line_test();
    parseStatement();
    parseStatementListPrime();
}
void parseStatementListPrime()
{
    new_line_test();
    Token token = peekNextToken(0);
    if (token.type == 10 || token.type == 9 || token.type == 8 || token.type == 4) // variable, write, read, if
    {
        parseStatement();
        parseStatementListPrime();
    }
}
void parseStatement()
{
    new_line_test();
    Token token = peekNextToken(0);
    if (token.type == 10) // assign
    {
        // check if the variable is in the table
        parseVariable(0);
        match(20); // assign
        parseExpression();
        match(23);
    }
    else if (token.type == 8) // read
    {
        getNextToken();
        fprintf(output, "%s %d\n", token.value, token.type);
        match(21); // LPAREN
        parseVariable(0);
        match(22); // RPAREN
        match(23); // SEMICOLON
    }
    else if (token.type == 9) // write
    {
        getNextToken();
        fprintf(output, "%s %d\n", token.value, token.type);
        match(21); // LPAREN
        parseVariable(0);
        match(22); // RPAREN
```

```
            match(23); // SEMICOLON
        }
        else if (token.type == 4) // if
        {
            getNextToken();
            fprintf(output, "%s %d\n", token.value, token.type);
            parseConditionExpression();
            new_line_test();
            match(5); // then
            parseStatement();
            new_line_test();
            match(6); // else
            parseStatement();
        }
        else
        {
            char errorMsg[128];
            sprintf(errorMsg, "***%d: Expected token: \"read\" / \"write\" / \"if\" / \"(variable)\" but got \"%s\".",
token.line,token.value);
            syntaxError(errorMsg, -1);
        }
}
void parseExpression()
{
    new_line_test();
    parseTerm();
    parseExpressionPrime();
}
void parseExpressionPrime()
{
    new_line_test();
    if (peekNextToken(0).type == 18) // minus
    {
        match(18);
        parseTerm();
        parseExpressionPrime();
    }
}
void parseTerm()
{
    new_line_test();
    parseFactor();
    parseTermPrime();
}
void parseTermPrime()
{
    new_line_test();
```

```
    if (peekNextToken(0).type == 19) // times
    {
        match(19);
        parseFactor();
        parseTermPrime();
    }
}
void parseFactor()
{
    new_line_test();
    Token token = peekNextToken(0);
    if (token.type == 10) // variable
    {
        parseVariable(0);
        Token token = peekNextToken(0);
        if (strcmp(token.value, "(") == 0)
        {                   // function call
            match(21); //(
            parseExpression();
            match(22); //)
        }
    }
    else if (token.type == 11) // number
    {
        getNextToken();
        fprintf(output, "%s %d\n", token.value, token.type);
    }
    else
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Unexpected token in factor", token.line);
        syntaxError(errorMsg, -1);
    }
}
void parseConditionExpression()
{
    new_line_test();
    parseExpression();
    parseRelationOperator();
    parseExpression();
}
void parseRelationOperator()
{
    new_line_test();
```

```
    Token token = getNextToken();
    if (token.type != 12 && token.type != 13 && token.type != 14 && token.type != 15 && token.type != 16
&& token.type != 17)
    {
        char errorMsg[128];
        sprintf(errorMsg, "***%d: Expected relation operator", token.line);
        syntaxError(errorMsg, -1);
    }
    else
    {
        fprintf(output, "%s %d\n", token.value, token.type);
    }
}
void read_tokens() // read all tokens to tokens array
{
    file = fopen("E:\\C++\\Compile\\output.dyd", "r");
    err = fopen("E:\\C++\\Compile\\output.err", "a");
    output = fopen("E:\\C++\\Compile\\output.dys", "w");
    pro = fopen("E:\\c++\\Compile\\output.pro ", " w ");
    var = fopen("E:\\c++\\Compile\\output.var ", " w ");

    fseek(err, 0, SEEK_END); // move the file pointer to the end of the file
    long size = ftell(err);   // get the position of the file pointer
    fseek(err, 0, SEEK_SET); // move the file pointer back to the start of the file

    if (size > 0)
    { // if the position of the file pointer is greater than 0, the file is not empty
        close_file();
        printf("There are errors in earlier stage.\n");
        system("pause");
        exit(1); // exit the program
    }
    // else continue with the program
    if (!file || !err || !output || !pro || !var)
    {
        printf("Error opening file.\n");
        exit(1);
    }
    char buffer[16];
    int length = 0;
    int ch = fgetc(file);
    while (ch != EOF)
    {
        length = 0;
        while (!isdigit(ch) || length == 0 || buffer[length - 1] != ' ') // read value from file
        {
            buffer[length++] = ch;
```

```
            ch = fgetc(file);
        }
        ungetc(ch, file);
        buffer[length - 1] = '\0';
        strcpy(tokens[tokenCount].value, buffer);
        fscanf(file, "%d", &tokens[tokenCount].type); // read type from file
        tokens[tokenCount].line = tokenLine;
        if (tokens[tokenCount].type == 24)
        { // end of line
            tokenLine++;
        }
        tokenCount++;
        fgetc(file); // handle new line character
        ch = fgetc(file);
    }
}
void print_tables() // print process table and variable table to correspond file
{
    for (int i = 1; i < varCount; i++)
    {
        fprintf(var, "Name: %s, Proc: %s, Kind: %d, Type: %s, Level: %d, Addr: %d\n", varTable[i].vname,
varTable[i].vproc, varTable[i].vkind, varTable[i].vtype, varTable[i].vlev, varTable[i].vadr);
    }

    for (int i = 1; i < funcCount; i++)
    {
        fprintf(pro, "Name: %s, Type: %s, Level: %d, First Addr: %d, Last Addr: %d\n", funcTable[i].pname,
funcTable[i].ptype, funcTable[i].plev, funcTable[i].fadr, funcTable[i].ladr);
    }
}
int main()
{
    read_tokens(); // read tokens from file to tokens array
    parseProgram();
    print_tables();
    close_file();
    system("pause");
    return 0;
}
```