《编译原理》实验报告

学　　院　计算机工程与科学学院

组　　号　10 组

实验题号　实验二

日　　期　2025 年 05 月 14 日

学号	姓名	主要工作	贡献因子
22122792	王潇	实验代码与报告	0.25
22121037	钱鹏程	实验代码	0.25
22121639	何勇乐	实验代码与报告	0.25
22121630	汪江豪	实验代码	0.25

《编译原理》实验报告

一、 实验题目

编写一个 PL/0 语言的词法分析程序，读取源程序文件并识别其中的关键字、标识符、常数、运算符和界符等单词，输出统一格式的二元式，源程序中的字符不区分大小写。

二、 实验内容设计与实现

2.1. 分词器

`getsym()` 函数是整个词法分析器的核心模块，负责从输入流中逐字符读取并识别出具有语义的记号（token）。该函数的设计体现了良好的状态控制、字符处理和错误检测能力，其算法逻辑不仅结构清晰，而且充分考虑了多种语言符号的识别规则与边界情况。

`getsym()` 首先调用代码1 `skip_whitespace()` 函数跳过所有空白字符，包括空格、制表符、回车符以及换行符，并在遇到换行符时自动更新当前的行号计数器 `lineNumber` ，为后续可能的语法分析或错误提示提供了重要的上下文信息。接着，函数根据当前字符的类型进行分类处理：如代码2如果字符是字母，则开始收集标识符或关键字；如代码3如果是数字，则尝试解析为整型数值，并在此过程中判断是否出现了非法标识符（如以数字开头但后跟字母的情况）；如果是特殊符号，则进入一个多层嵌套的 switch 判断结构，用于识别诸如 `:=、<=、>=、<>` 等多字符操作符，这种处理方式模拟了有限状态机的思想，使得多个字符组合能够被准确地识别为一个整体。

代码 1　skip_whitespace 函数

```
1.  void skip_whitespace() {
2.      char ch;
3.      while (
4.          (ch = inputFile.get()) == ' ' || ch == '\r' || ch == '\t'
    || ch == '\n')
5.      {
6.          if (ch == '\n') {
7.              lineNumber++;
8.          }
9.      }
10.
11.     inputFile.unget(); // 当前字符放回输入流
12. }
```

代码 2　getsym 函数

```
1.  void getsym() {
2.    if (isalpha(ch)) { // 以字母开头，可能是标识符或保留字
3.        int i    = 0;
4.        id[i++] = ch;
5.        while (isalnum((ch = inputFile.get()))) {
6.            id[i++] = ch;
7.        }
8.        id[i] = '\0';
9.        inputFile.unget(); // 将最后一个非字母数字字符放回输入流
10.
11.       if (strcmp(id, "begin") == 0)
12.           sym = beginsym;
13.       else if (strcmp(id, "end") == 0)
14.           sym = endsym;
15.       else if (strcmp(id, "if") == 0)
16.           sym = ifsym;
17.       else if (strcmp(id, "then") == 0)
18.           sym = thensym;
19.       else if (strcmp(id, "while") == 0)
20.           sym = whilesym;
21.       else if (strcmp(id, "write") == 0)
22.           sym = writesym;
23.       else if (strcmp(id, "read") == 0)
24.           sym = readsym;
25.       else if (strcmp(id, "do") == 0)
26.           sym = dosym;
27.       else if (strcmp(id, "call") == 0)
28.           sym = callsym;
29.       else if (strcmp(id, "const") == 0)
30.           sym = constsym;
31.       else if (strcmp(id, "var") == 0)
32.           sym = varsym;
33.       else if (strcmp(id, "procedure") == 0)
34.           sym = procsym;
35.       else
36.           sym = ident; // 如果不是保留字，则是标识符
37.   }
```

代码 3　getsym 函数

```
1.   else if (isdigit(ch)) { // 以数字开头，可能是数字或者不合法标识符
2.       int i = 0;
3.       num   = 0;
4.       do {
5.           num     = num * 10 + (ch - '0');
6.           id[i++] = ch;
7.           ch      = inputFile.get();
8.       } while (isdigit(ch));
9.       while (isalpha(ch)) {
10.          Invalid_identifier = true;
11.          id[i++]            = ch;
12.          ch                 = inputFile.get();
13.          sym                = ident;
14.      }
15.      id[i] = '\0';
16.      if (Invalid_identifier) {
17.          std::cout << "Error: Invalid identifier" << " at line
"
18.                    << lineNumber << "\n";
19.      } else {
20.          inputFile.unget(); // 将最后一个非数字且非字母字符放回输入流
21.          sym = number;
22.      }
23.  }
```

为了保证输入流的完整性与一致性，程序使用了 `unget()` 方法将已经读取但尚未处理的字符重新放回输入流中，这种回溯机制确保了每次调用 `getsym()` 时都能正确地从上一次未处理完的位置继续读取，避免了字符的遗漏或重复处理。此外，在识别关键字时，程序通过逐一比较字符串的方式判断其是否为保留字，否则作为普通标识符返回，这种方式虽然简单，但在小型语言系统中具备足够的效率和可维护性。 函数还集成了基本的错误检测功能。例如，在识别数字时若发现后面紧跟字母，则将其标记为非法标识符并在输出时予以提示；对于无法识别的字符，程序会将其暂存到 `unknownBuffer` 中，并在最终输出时给出明确的错误信息，指出非法字符及其出现的行号，从而提升了程序的健壮性和调试友好性。

三、 实验数据测试

3.1. 简单正例

输入文件如代码4。

代码 4　输入正例

```
const a = 10;
var b, c;
begin
    read(b);
    c := a + b;
    write(c);
end.
```

输出如下。

```
(constsym, const)
(ident, a)
(eql, =)
(number, 10)
(semicolon, ;)
(varsym, var)
(ident, b)
(comma, ,)
(ident, c)
(semicolon, ;)
(beginsym, begin)
(readsym, read)
(lparen, ( )
(ident, b)
(rparen, ))
(semicolon, ;)
(ident, c)
(becomes, :=)
(ident, a)
(plus, +)
(ident, b)
(semicolon, ;)
(writesym, write)
(lparen, ( )
(ident, c)
(rparen, ))
(semicolon, ;)
(endsym, end)
(period, .)

```

3.2. 下划线测试

输入文件如代码5。

代码 5　下划线测试

```
var a1,b1;
var a_1,b_1;
begin
     a1=2;
     b1=3;
     a_1=4;
     b_1=5;
end.
```

报错如图1。

```
Error: Unknown symbol '_' at line 2
Error: Unknown symbol '_' at line 2
Error: Unknown symbol '_' at line 6
Error: Unknown symbol '_' at line 7
请按任意键继续. . .
```

图 1　下划线报错

输出如下。

```
(varsym, var)
(ident, a1)
(comma, ,)
(ident, b1)
(semicolon, ;)
(varsym, var)
(ident, a)
(Unknown symbol, _)
(number, 1)
(comma, ,)
(ident, b)
(Unknown symbol, _)
(number, 1)
(semicolon, ;)
(beginsym, begin)
(ident, a1)
(eql, =)
(number, 2)
(semicolon, ;)
(ident, b1)
(eql, =)
(number, 3)
(semicolon, ;)
(ident, a)
(Unknown symbol, _)
(number, 1)
(eql, =)
(number, 4)
(semicolon, ;)
(ident, b)
(Unknown symbol, _)
(number, 1)
(eql, =)
(number, 5)
(semicolon, ;)
(endsym, end)
(period, .)
```

3.3. 括号测试

输入文件如代码6。

代码 6　存在括号

```
var a1,b1;
begin
    a1=2;
    b1=3;
    a1:=a1+(b1*a1);
end.
```

输出如下。

```
(varsym, var)
(ident, a1)
(comma, ,)
(ident, b1)
(semicolon, ;)
(beginsym, begin)
(ident, a1)
(eql, =)
(number, 2)
(semicolon, ;)
(ident, b1)
(eql, =)
(number, 3)
(semicolon, ;)
(ident, a1)
(becomes, :=)
(ident, a1)
(plus, +)
(lparen, ( )
(ident, b1)
(times, *)
(ident, a1)
(rparen, ))
(semicolon, ;)
(endsym, end)
(period, .)
```

3.4. ++测试

输入文件如代码7。

代码 7　存在++

```
var a1,b1;
begin
    a1=2;
    b1=3;
    a1:=a1++;
end.
```

输出如下。

```
(varsym, var)
(ident, a1)
(comma, ,)
(ident, b1)
(semicolon, ;)
(beginsym, begin)
(ident, a1)
(eql, =)
(number, 2)
(semicolon, ;)
(ident, b1)
(eql, =)
(number, 3)
(semicolon, ;)
(ident, a1)
(becomes, :=)
(ident, a1)
(plus, +)
(plus, +)
(semicolon, ;)
(endsym, end)
(period, .)
```

3.5. 错误的标识符

输入文件如代码8。

代码 8　错误的标识符

```
var 123abc,a2;
begin
    a2=2;
    123abc=3;
    a2:=123abc+a2;
end.
```

报错如图2。

```
Error: Invalid identifier at line 1
Error: Invalid identifier at line 4
Error: Invalid identifier at line 5
请按任意键继续. . .
```

图 2　错误的标识符

3.6. 非法字符

输入如下，输入了‘@’这个非法字符。

代码 9　非法字符

```
1. var a1,b1;
2. begin
3.     a1=2;
4.     b1=3;
5.     a1:=a1@b1;
6. end.
```

报错如图3。

```
Error: Unknown symbol '@' at line 5
请按任意键继续. . .
```

图 3　非法字符

四、 实验总结

本次实验的主要任务是实现一个简单的词法分析器，能够识别 PL/0 语言中的关键字、标识符、常数、运算符和界符等单词，并输出统一格式的二元式。通过对 `getsym()` 函数的设计与实现，我们深入理解了词法分析的基本原理和方法。通过本次实验，我们不仅掌握了词法分析器的基本实现方法，还提高了对编译原理中词法分析部分的理解和应用能力。同时，我们也认识到在实际编程中，良好的代码结构和错误处理机制是非常重要的，这将为我们今后的学习和工作打下坚实的基础。