实验一：

一、 **实验目的:** 给出 PL/0 语言词法规范，编写 PL/0 语言的词法分析程序。

二、 **实验准备:** 安装好 C 语言或 C++

三、 **实验时间:2024 年 10 月 20 日星期日**

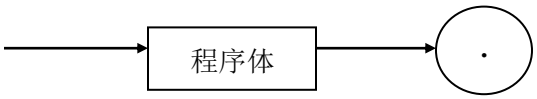
四、 **实验内容:** 给出 PL/0 语言词法规范，编写 PL/0 语言的词法分析程序。

五、 **基本原理:** 程序中先判断这个句语句中每个单元为关键字、常数、运算符、界符，对与不同的单词符号给出不同编码形式的编码，用以区分之。

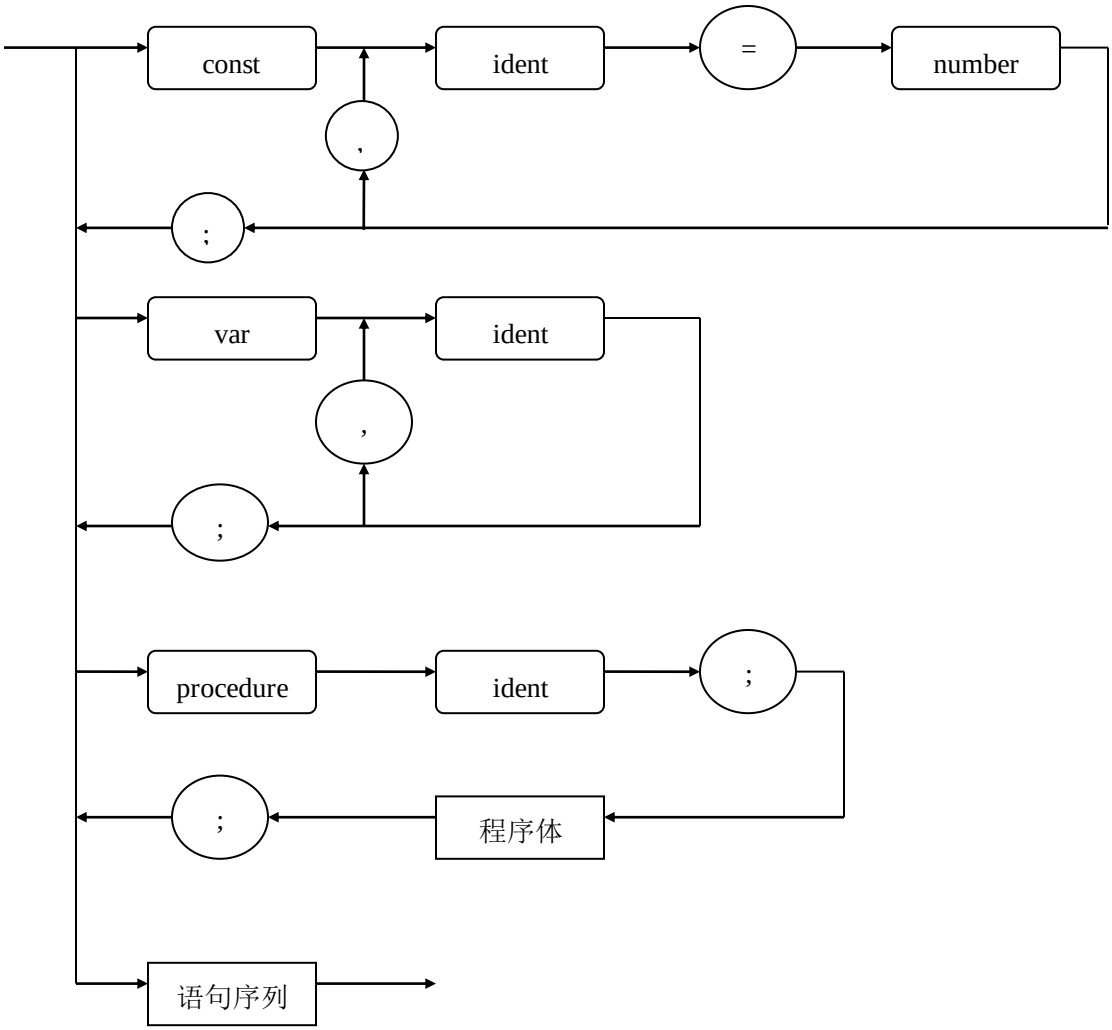
六、 **实现过程**

流程图

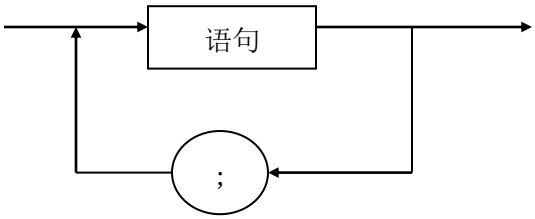
程序



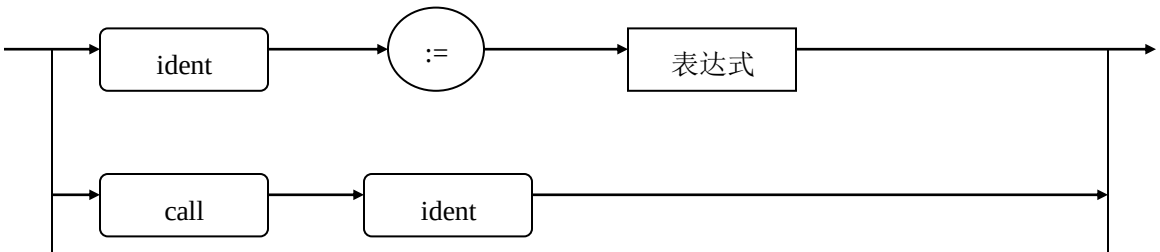
程序体



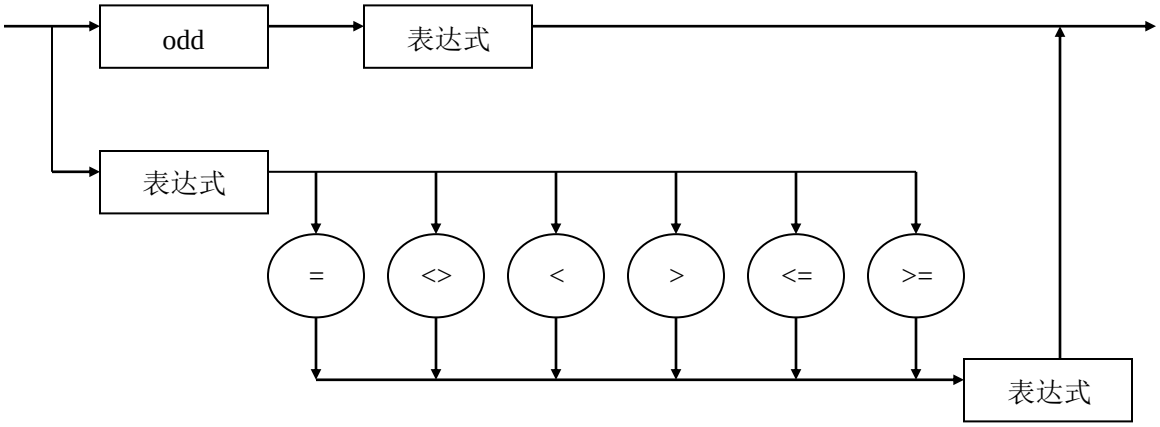
语句序列



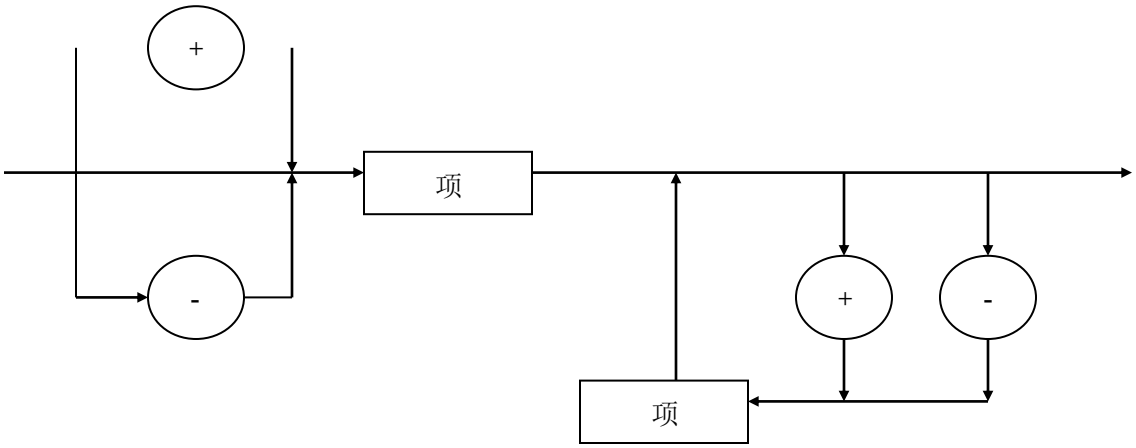
语句



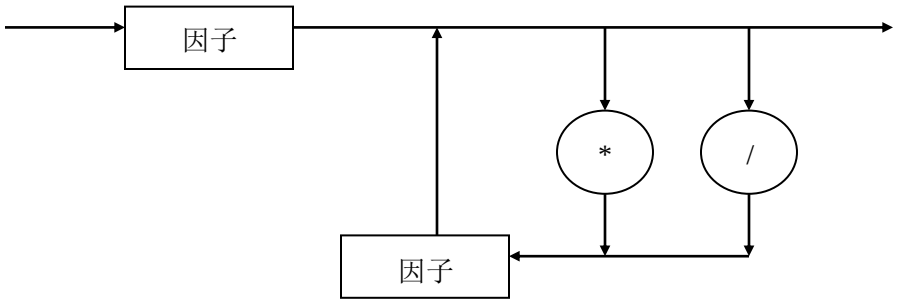
条件



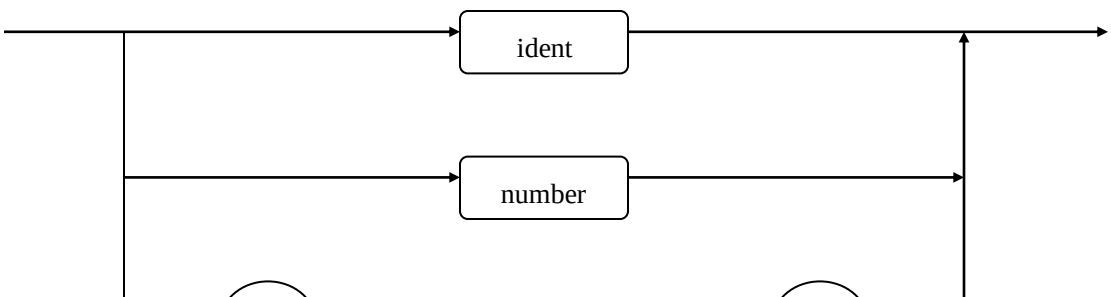
表达式



项



因子



1. 核心代码:

```
#include <bits/stdc++.h>
using namespace std;

#define SYS_IDENT     10086
#define SYS_NUMBER    10087

map<string, int> MAP;
FILE *out;
const vector<string>SYS_STRING={

"const","end","procedure","if","then","while","do","begin","call","var","odd","+","-","*","/","(",")
","=",",",".",";",">=","<=","==","{","}",":=","<>"
};
const vector<int>SYS_NUM={
    1,2,3,4,5,6,7,8,9,10,11,12,13 ,14 ,15 ,16,17,18,19,20,21,22,23,24,25,26,27,28
};
void CreateMap()
{
    for(int i=0;i<SYS_STRING.size();i++)
    {
        MAP[SYS_STRING[i]]=SYS_NUM[i];
    }
}
void write(int num, string x)
{
    printf("(%-5d,%-10s)\n", num, x.c_str());
    fprintf(out, "(%-5d,%-10s)\n", num, x.c_str());
}
inline string tr(char s)
{
    return string(1, s);
}
void compile(FILE *FilePtr)
{
    int num = 0;
    string name;
    char CharGot;
    CharGot = fgetc(FilePtr);
```

```
while (CharGot != EOF)
{
    while (CharGot == '\n' || CharGot == '\t' || CharGot == '\r' || CharGot == ' ')
    {
        CharGot = fgetc(FilePtr);
    }
    if (isdigit(CharGot))
    {
        while (isdigit(CharGot))
        {
            num = num * 10 + CharGot - '0';
            CharGot = fgetc(FilePtr);
        }

        write(SYS_NUMBER, string(to_string(num)));
        /*
            对数字 num 进行处理
        */
        num = 0;
    }
    else if (isalpha(CharGot))
    {
        string name = tr(CharGot);
        CharGot = fgetc(FilePtr);
        while (isdigit(CharGot) || isalpha(CharGot))
        {
            name += CharGot;
            CharGot = fgetc(FilePtr);
        }
        if (MAP.find(name) != MAP.end())
        {
            write(MAP[name], name); // 如果是关键字
        }
        else
        {
            write(SYS_IDENT, name); //   如果是声明变量
        }
    }
    else
    {
        if (CharGot == ':')
        {
            CharGot = fgetc(FilePtr);
            if (CharGot == '=')
```

```
                {
                    write(MAP[":="], ":=");
                    CharGot = fgetc(FilePtr);
                }
                else
                {
                    printf("\":= \" error!\n");
                }
            }
            else if (CharGot == '+' || CharGot == '-' || CharGot == '*' || CharGot == '/' || CharGot
== '(' || CharGot == ')' || CharGot == ',' || CharGot == '.' || CharGot == ';')
            {
                write(MAP[tr(CharGot)], tr(CharGot));
                CharGot = fgetc(FilePtr);
            }
            else if (CharGot == '<')
            {
                CharGot = fgetc(FilePtr);
                if (CharGot == '>')
                {
                    write(MAP["<>"], "<>");
                    CharGot = fgetc(FilePtr);
                }
                else if (CharGot == '=')
                {
                    write(MAP["<="], "<=");
                    CharGot = fgetc(FilePtr);
                }
                else
                {
                    write(MAP["<"], "<");
                }
            }
            else if (CharGot == '>')
            {
                CharGot = fgetc(FilePtr);
                if (CharGot == '=')
                {
                    write(MAP[">="], ">=");
                    CharGot = fgetc(FilePtr);
                }
                else
                {
                    write(MAP[">"], ">");
```

```
                }
            }
            else if (CharGot == '=')
            {
                CharGot = fgetc(FilePtr);
                if (CharGot == '=')
                {
                    write(MAP["=="], "==");
                    CharGot = fgetc(FilePtr);
                }
                else
                {
                    write(MAP["="], "=");
                }
            }
            else if (CharGot == '{')
            {
                while (CharGot != '}')
                    CharGot = fgetc(FilePtr);
                CharGot = fgetc(FilePtr);
            }
        }
    }
}
int main()
{
    CreateMap();

    FILE *ptr = fopen("test.pl", "r");
    out = fopen("out.pl", "w");

    compile(ptr);
    return 0;
}
```

2. 运行结果:

导入文本:

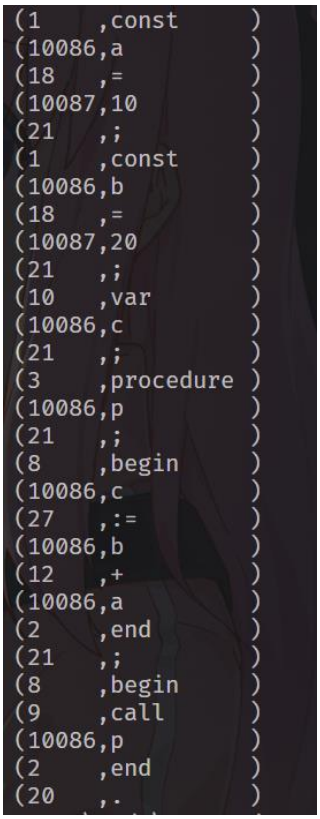
```
const a=10;     {常量声明}
const b=20;
var c;          {变量声明}
procedure p;    {过程声明}
    begin
        c:=b+a
    end;
```

```
begin
    call p
end.
```

输出:

```
(1      ,const      )
(10086,a            )
(18     ,=          )
(10087,10           )
(21     ,;          )
(1      ,const      )
(10086,b            )
(18     ,=          )
(10087,20           )
(21     ,;          )
(10     ,var        )
(10086,c            )
(21     ,;          )
(3      ,procedure )
(10086,p            )
(21     ,;          )
(8      ,begin      )
(10086,c            )
(27     ,:=         )
(10086,b            )
(12     ,+          )
(10086,a            )
(2      ,end        )
(21     ,;          )
(8      ,begin      )
(9      ,call      )
(10086,p            )
(2      ,end        )
(20     ,.          )
```

3. 运行截图



七、 实验总结

1. 实验过程分析:

- 识别到 token 就直接输出了;
- C 库函数 int isalpha(int c)：判断字符是否是字母，int isdigit (int c)：判断字符是否是数字。当然也直接用 ASCII 码。

种别码及其对应类型

const	1
end	2
procedure	3
if	4
then	5
while	6
do	7
begin	8
call	9
var	10
odd	11
+	12
-	13
*	14
/	15
(	16

)	17
=	18
,	19
.	20
;	21
>=	22
<=	23
==	24
{	25
}	26
:=	27
<>	28
命名词	10086
数字	10087

2. 心得体会:

词法分析是编译器设计过程中的第一步，其任务是将输入的源代码字符串分解为一系列有意义的符号，并为每个词法单元赋予相应的类别（即词法类型）。本次词法分析程序的设计与实现实验是一次非常有价值的学习经历，它不仅让我对词法分析的核心知识有了更深的理解，也为后续学习语法分析、语义分析等高级编译器设计内容打下了坚实的基础。