编译原理大作业报告

PL/0 语言语法分析程序

学号：517021911065　姓名：刘浩文　班级：F1703603
联系方式：IssacLiewX@sjtu.edu.cn

2019 年 12 月 16 日

摘要

在这次大作业实践中，我完成了大作业的全部要求，包括：将词法分析部分添加进语法分析部分中，并修改了原来词法分析中对于语法分析不合理的地方；补全了对单链表进行操作的函数，清楚了单链表在语法分析部分中的功能；结合流程图阅读了 pl0.c 文件，了解了各函数功能和基本流程；补全了参与语法分析的 $block()$、$vardeclaration()$、$constdeclaration()$ 等几个函数；对程序的其他地方进行了一些改进完善。之后，使用完成的语法分析程序对提供的 PL/0 源程序进行了编译，得到了正确输出结果。此外，我对提供的 PL/0 源程序进行了一些修改，测试了该语法分析程序的检错能力，也得到了较好的结果。我还使用 PL/0 语言编写了一个计算正整数 a 和 b 的 $Bezout$ 等式的程序（在 $Bezout.txt$ 文件中），并使用语法分析程序对其进行了编译，也得到了正确的输出结果。本报告先概述了使用递归下降法的语法分析的功能和原理，然后对任务要求的完成情况及最后的输出结果进行了展示，也展示了对程序其他地方的一些完善，之后记录并分析了实践过程中出现的一些问题，最后表达了对本次作业的感想与一些建议。

关键词：　编译原理　　语法分析程序　　PL/0 语言

1 语法分析过程描述

1.1 语法分析概述

语法分析程序的功能是以词法分析器生成的单词符号序列作为输入，根据语言的语法规则（描述程序语言的上下文无关文法），识别出各种语法成分，并在分析过程中进行语法检查，检查所给单词符号序列是否是该语言文法的一个句子。若是，则以该句子的某种形式的语法树作为输出；若不是，则表明有错误，并指出错误的性质和位置。而本次大作业的语法分析程序只实现了对源程序进行语法检错及报错的功能。

在语法分析方法中，递归下降分析法是确定的自上而下分析法，它要求文法是 $LL(1)$ 文法。其基本思想是：对文法中的每个非终结符编写一个函数，每个函数的功能是识别由该非终结符所表示的语法成分。由于描述语言的文法常常是递归定义的，因此相应的这组函数必然以相互递归的方式进行调用，所以将此种分析法称为递归下降分析法。

1.2 对 PL/0 语法分析部分的认识与分析

该 PL/0 语言语法分析部分是基于词法分析部分对单词符号的正确识别。在源程序字符串未出现词法错误的前提下，语法分析部分将对词法分析部分得到的单词符号串进行语法分析，检查其是否有语法错误。

结合所给流程图（如图 1）可知，PL/0 语言主要有六个语法成分：程序体 ($block$)、语句 ($statement$)、条件 ($condition$)、表达式 ($expression$)、项 ($term$)、因子 ($factor$)，他们也是 PL/0 语言文法的六个非终结符。根据递归下降分析法的工作流程，编写对应的六个函数：$block()$、$statement()$、$condition()$、$expression()$、$term()$、$factor()$。这六个函数将相互递归调用，组成语法分析部分的基本结构。另外，还有其他两个语法成分：常量声明 ($constdeclaration$) 与变量声明 ($vardeclaration$)，常量声明的 $const$ 后只能接一个或多个常量定义赋值表达式，变量声明的 var 后只能接一个或多个标识符。

为了完成语法分析程序，仅靠提供的《要求》中的流程介绍及不全的源码似乎略显不足，于是我还在网上查找了 PL/0 文法相关的资料。结合老师提供的介绍与源码和自己的理解，总结出 PL/0 语言的文法如下：

$$
\begin{aligned}
&program && \longrightarrow block.\\
&block && \longrightarrow [constdeclaration][vardeclaration][procdeclaration]\ \ statement\\
&constdeclaration && \longrightarrow \textbf{const}\ \ constdefine\{,constdefine\};\\
&constdefine && \longrightarrow \textbf{ident} = \textbf{number}\\
&vardeclaration && \longrightarrow \textbf{var}\ \ \textbf{ident}\{,\textbf{ident}\};\\
&procdeclaration && \longrightarrow \textbf{procedure}\ \ \textbf{ident};block;\{\textbf{procedure}\ \ \textbf{ident};block;\}\\
&statement && \longrightarrow \textbf{ident} := expression|\textbf{call}\ \ \textbf{ident}|\textbf{begin}\ \ statement\{;statement\}\ \ \textbf{end}\\
& && |\textbf{if}\ \ condition\ \ \textbf{then}\ \ statement|\textbf{while}\ \ condition\ \ \textbf{do}\ \ statement|\epsilon\\
&condition && \longrightarrow expression\ \ relop\ \ expression\\
&relop && \longrightarrow = \mid <> \mid < \mid > \mid <= \mid >=\\
&expression && \longrightarrow [+|-]term\{+term|-term\}\\
&term && \longrightarrow factor\{*factor|/factor\}\\
&factor && \longrightarrow \textbf{ident}|\textbf{number}|(expression)
\end{aligned}
$$

据此，可以依照递归下降文法清晰地写出其语法分析程序。其他具体的细节将在语法程序的补全与改进中说明。

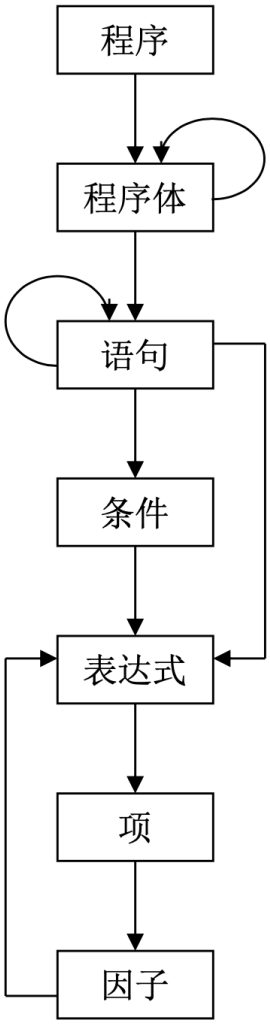

图 1: 提供的流程图

2 语法分析程序的补全与改进

2.1 调试环境

以下列出本次实践的基本调试环境:

- **机器**: $MacBook\ Pro\ (13-inch,\ 2016,\ Four\ Thunderbolt\ 3\ Ports)$
- **系统**: $macOS\ Catalina\ Version 10.15.2$
- **调试软件**: $Xcode\ Version\ 11.3\ (11C29)$

提前说明: 由于我的编程环境 Xcode 在链接源文件时总是会出现问题，set.c 总是会出现冲突，查找网上的资料也不能解决，还耗费了大量的时间。考虑到编译原理大作业的目的不是训练编程或调试环境，而是通过完成语法分析的功能来加深对知识的理解，所以我干脆将 set.c 的函数合并到了 pl0.c 中，不影响程序的运行，希望老师谅解。

2.2 语法分析程序的补全

Step 1: 阅读 *set.h* 和 *set.c*，*symset* 数据结构。根据已经提供的部分代码，完成 *void setinset*(*symset s*, *int elem*) 函数（该函数功能为将一个新元素按 *elem* 大小插入对应位置）；完成 *int inset*(*int elem*, *symset s*) 函数（该函数功能为查找一个元素是否在 *symset* 中，若在返回 1，不在返回 0）。

阅读 *set.h* 文件可知，*symset* 数据结构是一个单链表数据结构，则 *set.c* 中的各函数按单链表的特点设计即可。

```
void setinsert(symset s, int elem)//该函数将一个新结点（数据为elem）按数据大小插入链表s中对应位置
{
    snode* p;
    snode* ss = s;
    p = (snode*) malloc(sizeof(snode));
    p->elem = elem;                    //这里新建一个结点，数据为elem
    s = s->next;                       //s指向s的下一个结点
    while((s != NULL) && (s->elem <= elem))
    {
        ss = s;
        s = s->next;
    }//为新结点找到合适的位置
    p->next = s;
    ss->next = p;
} // setinsert
```

```
int inset(int elem, symset s)//该函数功能为查找一个元素是否在symset中，若在返回1，不在则返回0
{
    while((s != NULL) && (elem != s->elem))
        s = s->next;

    if (s != NULL) return 1;
    else return 0;
} // inset
```

Step 2: 阅读 *pl0.c*，首先将词法分析大作业中添加部分自行添加到空缺部分。然后了解各函数功能和基本流程，尤其注意结合流程图进行阅读。在报告中将流程进行叙述，有收获的点可以谈谈感想。

关于流程的叙述在 1.2 对 PL/0 语法分析部分的认识与分析一节中已有叙述，这里不再赘述。可以看到，本语法分析是依照 PL/0 语言的文法进行递归下降分析，以非终结符为函数名相互递归调用。另外在本程序中，单链表的作用是存放当前语法成分的预测识别单词符号集合。

另外对于词法分析部分，为了使其适配语法分析，而对其在注释的跳过的流程做了一些修改，主要是删去了对注释的识别，因为注释不需要被识别，只需要被跳过，否则会对语法分析造成影响：

```
    /* TO BE MODIFIED */
    /* Skip  Notes*/
    else if (ch == '(')
    {
        g_etch();
        if (ch == '*')
        {
            addr = cx - 1;//记录注释起始位置，方便报错
            g_etch();
            char ch1 = ch;//ch1与ch保存最近经过的两个符号
            while((ch1 != '*') || (ch != ')') )//检查最近经过的两个符号是否组成注释结束符
            {
```

```
                ch1 = ch;
                g_etch();
                if(feof(infile))
                {
                    printf("\nPROGRAM INCOMPLETE\n");
                    printf("ERROR: Line %d Unterminated Notes.\n",addr);
                    exit(1);
                    //若直到程序结束都没有注释结束符，则报错，指出注释起始位置
                }
            }//这里跳过所有注释开始符之后的字符串，直到遇到注释结束符
            g_etch();
            getsym();
        }
        else sym = SYM_LPAREN;
    }
```

Step 3: 完成 *void block*() 函数中的 *constant declarations* 和 *variable declarations* 两部分处理。

对 *constant declarations* 的处理依据文法规则：

$$\begin{aligned} &constdeclaration && \longrightarrow \textbf{const}\ \ constdefine\{,constdefine\}; \\ &constdefine && \longrightarrow \textbf{ident} = \textbf{number} \end{aligned}$$

对 *variable declarations* 的处理依据文法规则：

$$vardeclaration \longrightarrow \textbf{var}\ \ \textbf{ident}\{,\textbf{ident}\};$$

具体流程图如图 2所示，左边为对 *constant declarations* 的处理，右边为对 *variable declarations* 的处理。依据该流程图可得到两部分的代码。

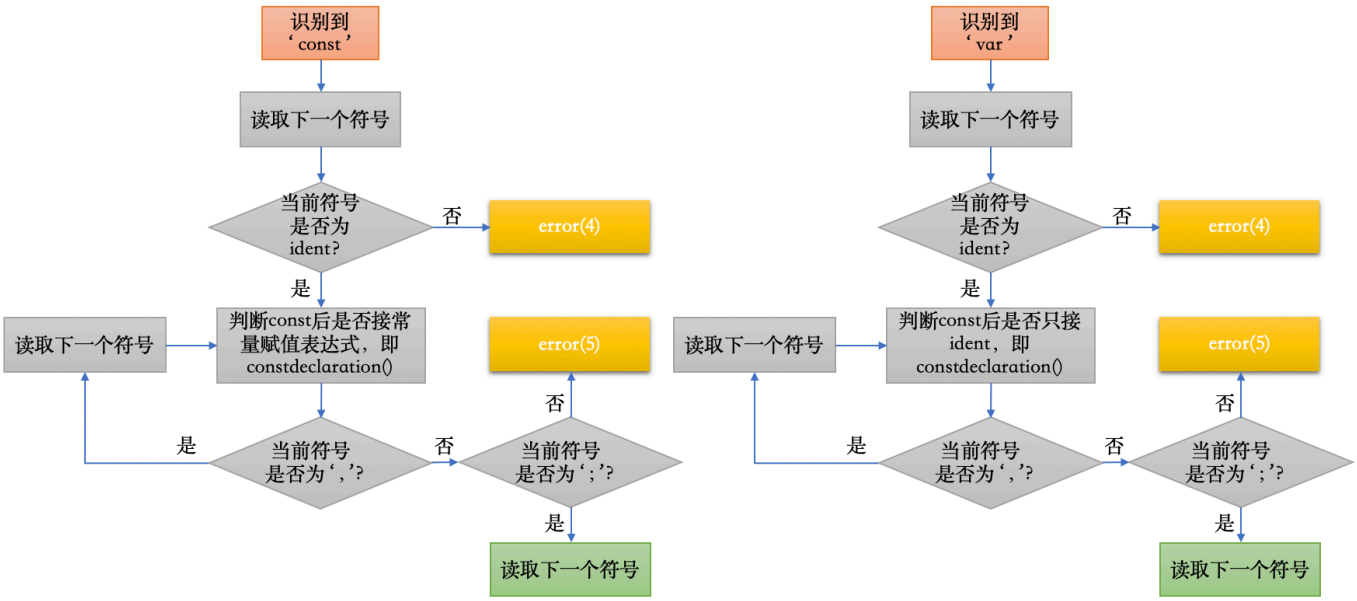

图 2: 对 *constant declarations* 及 *variable declarations* 处理的流程图

```
    if (sym == SYM_CONST)
    { // constant declarations
            getsym();
            if (sym == SYM_IDENTIFIER)
            {
                constdeclaration();
                while (sym == SYM_COMMA)
                {
                    getsym();
                    constdeclaration();
                }
                if (sym == SYM_SEMICOLON)
                    getsym();
                else error(5);
            }
            else error(4);
    } // if
```

```

    if (sym == SYM_VAR)
    { // variable declarations
            getsym();
            if (sym == SYM_IDENTIFIER)
            {
                vardeclaration();
                while (sym == SYM_COMMA)
                {
                    getsym();
                    vardeclaration();
                }
                if (sym == SYM_SEMICOLON)
                    getsym();
                else error(5);
            }
            else error(4);
    } // if
```

Step 4: 参照 *void vardeclaration*()，完成 *void constdeclaration*() 函数。

结合 *void vardeclaration*() 函数，从之前对 *constant declarations* 及 *variable declarations* 处理的补充可知，*void constdeclaration*() 函数是为了完成对 *constant declarations* 处理中的对 *const* 关键字后接常量赋值表达式的判断与识别。因此在 *void constdeclaration*() 函数中只要完成对完整常量声明赋值表达式的识别即可。再结合 *plo.h* 中报错的种类，可以看到其中 *error*(1) 是：*"Found ':=' when expecting '=' ."*，为了充分利用报错且使报错更为精准，特别加入对 ':=' 的识别，因为根据文法，PL/0 的常量声明赋值表达式中的赋值符号只能为 '=' 而不能为 ':='。设计 *void constdeclaration*() 函数的流程图如图 3，根据流程图可以完成 *void constdeclaration*() 函数的代码。且设计完 *void constdeclaration*() 函数后，发现原 *void vardeclaration*() 函数也有不足，所以对 *void vardeclaration*() 函数也进行了修改，将在 *step* 5 中说明。

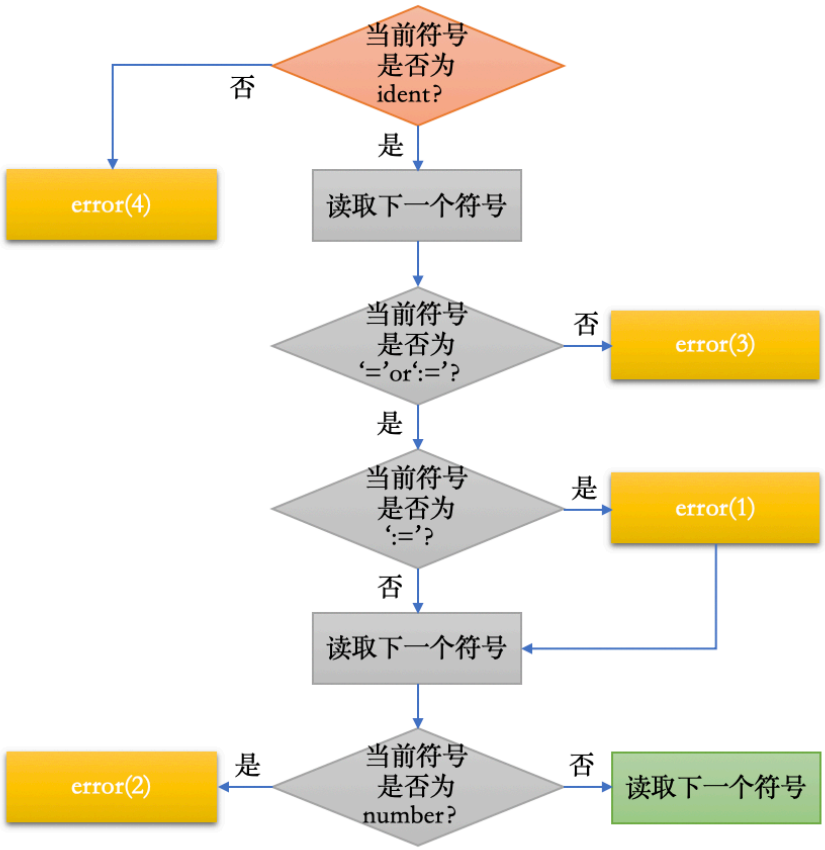

图 3: 设计 *void constdeclaration*() 函数的流程图

```
void constdeclaration(void)//这个函数检测const语句语法是否正确，即const后要接赋值表达式。
{
    if (sym == SYM_IDENTIFIER)
    {
        getsym();
        if ((sym == SYM_EQU) || (sym == SYM_BECOMES))
        {
            if (sym == SYM_BECOMES)
                error(1);
            getsym();
            if (sym == SYM_NUMBER)
                getsym();
            else error(2);
        }
        else error(3);
    }
    else error(4); // There must be an identifier to follow 'const', 'var', or 'procedure'.
} // constdeclaration
```

Step 5: **自行调试和完善。**

从 PL/0 语言的文法可知，变量声明，即关键字 **var** 后只能接标识符而不能像其他大多数高级语言那样在变量声明时能同时赋值。所以在语法分析变量声明时应考虑到这种情况。但是提供的源程序的 *void vardeclaration*() 函数却只是识别了紧接其后的标识符，没有继续识别同时给变量赋值这种容易犯的错

误；且修改测试的 PL/0 源程序后，可以发现如果不对 *void vardeclaration*() 函数进行修改的话，这种错误会给语法分析造成很大的影响。如图 4所示，在发现变量声明中出现标识符以外的赋值语句时原函数只能进行大概的报错，且不能跳过错误部分导致之后的语法分析完全不能进行下去。所以为了使语法分析程序更有效，我将 *void vardeclaration*() 函数修改成了如下形式，并增加了 *error*(26) 报错，通过有效地跳过错误部分使之后的语法分析能继续进行，且能精确报错，如图 5。

```
/* 26 */    "Vardeclaration wrong, after var should be only identifier but not expression."
```

```
void vardeclaration(void)//这个函数检测var语句语法是否正确，即var后要接一个变量标识符。
{
  if (sym == SYM_IDENTIFIER)
  {
    getsym();
        while((sym == SYM_EQU)||(sym == SYM_BECOMES))
        {
            error(26);
            getsym();
            if((sym == SYM_NUMBER) || (sym == SYM_IDENTIFIER))
                getsym();
        }
  }
  else
  {
    error(4); // There must be an identifier to follow 'const', 'var', or 'procedure'.
  }
} // vardeclaration
```

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85;
00001  var x=1,y,z,q,r;
            ^
Error   5: Missing ',' or ';'.
             ^
Error   7: Statement expected.
                ^
Error  13: ':=' expected.
                ^
Error  24: The symbol can not be as the beginning of an expression.
                  ^
Error  23: The symbol can not be followed by a factor.
                     ^
Error   8: Follow the statement is an incorrect symbol.
00002
00003  procedure multiply;   (*note1 *)
               ^
Error   9: '.' expected.
There are 7 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 4: 原来的程序对变量声明同时赋值的报错

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85;
00001  var x=1,y,z,q,r;
           ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
00009       if odd b then z := z + a;
00010       a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011   end
00012  end;
00013
00014  procedure divide;
00015  var w;
00016  begin
00017   r := x; q := 0; w := y;
00018   while w > y do
00019   begin
00020       q := 2 * q; w := w / 2;
```

图 5: 修改后的程序对变量声明同时赋值的报错

不仅如此，对变量声明同时连续赋值也能精确报错，且不会影响之后的语法分析。

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85;
00001  var x=a=b=c=d=1,y,z,q,r;
           ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
             ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
               ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
                 ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
                   ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
00009       if odd b then z := z + a;
00010       a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011   end
00012  end;
00013
00014  procedure divide;
```

图 6: 修改后的程序对变量声明同时连续赋值的报错

3 程序输出结果展示

3.1 对提供的 test.txt 中源程序进行语法分析

源程序展示如下。

```
const m = 7, n = 85;
var x,y,z,q,r;

procedure multiply;    (*note1 *)
var a,b;
begin
  a :=x;b :=y;z :=0;
  while b > 0 do
  begin
    if odd b then z := z + a;
    a := 2 * a; b := b / 2;      (*note5as1d6a*)
  end
end;

procedure divide;
var w;
begin
  r := x; q := 0; w := y;
  while w > y do
  begin
    q := 2 * q; w := w / 2;
    if w <= r then
    begin
      r := r - w;
      q := q + 1;
    end;
  end
end;

procedure gcd;
var f, g;
begin
  f := x;
  g := y;
  while f <> g do
  begin
    if f < g then g := g - f;
    if g < f then f := f - g;
  end
end;

procedure max;
var i, j, k;
begin
  i := x;
  j := y;
  k := 0;
  if i < j then k := j;
  if j < i then k := i;
end;


begin
  x := m; y := n; call multiply;
  x := 25; y := 3; call divide;
  x := 34; y := 36; call gcd;
end
```

58 .

使用完成的语法分析程序进行编译后输出结果如图 7、8。

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85;
00001  var x,y,z,q,r;
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
00009       if odd b then z := z + a;
00010       a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011   end
00012  end;
00013
00014  procedure divide;
00015  var w;
00016  begin
00017   r := x; q := 0; w := y;
00018   while w > y do
00019   begin
00020       q := 2 * q; w := w / 2;
00021       if w <= r then
00022       begin
00023           r := r - w;
00024           q := q + 1;
00025       end;
00026   end
00027  end;
00028
```

图 7: 语法分析 *test.txt* 输出结果

```
00029  procedure gcd;
00030  var f, g;
00031  begin
00032   f := x;
00033   g := y;
00034   while f <> g do
00035   begin
00036       if f < g then g := g - f;
00037       if g < f then f := f - g;
00038   end
00039  end;
00040
00041  procedure max;
00042  var i, j, k;
00043  begin
00044   i := x;
00045   j := y;
00046   k := 0;
00047   if i < j then k := j;
00048   if j < i then k := i;
00049  end;
00050
00051
00052  begin
00053   x := m; y := n; call multiply;
00054   x := 25; y := 3; call divide;
00055   x := 34; y := 36; call gcd;
00056  end
00057  .

   success with no error!
Compile End!
Program ended with exit code: 0
```

图 8: 语法分析 *test.txt* 输出结果

接下来进行报错测试。由于大作业要求中只让我们设计了有关常量声明与变量声明的语法分析，因此以下报错测试也主要围绕这两部分。

1. **测试常量声明的语法分析：**

- 测试 $error(1)$: "Found ′ :=′ when expecting ′ =′ .", 将第一行 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85;$ 改为 $const\ \ m\ \ :=\ \ 7,\ \ n\ \ =\ \ 85;$

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000   const m := 7, n = 85;
                 ^
Error    1: Found ':=' when expecting '='.
00001   var x,y,z,q,r;
00002
00003   procedure multiply;   (*note1 *)
00004   var a,b;
00005   begin
00006    a :=x;b :=y;z :=0;
00007    while b > 0 do
00008    begin
00009        if odd b then z := z + a;
00010        a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011    end
00012   end;
```

图 9: 测试 $error(1)$

```
00052   begin
00053    x := m; y := n; call multiply;
00054    x := 25; y := 3; call divide;
00055    x := 34; y := 36; call gcd;
00056   end
00057   .
There are 1 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 10: 测试 $error(1)$

- 测试 $error(2)$: "There must be a number to follow ′ =′ .",将第一行 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85;$ 改为 $const\ \ m\ \ =\ \ ,\ \ n\ \ =\ \ 85;$

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000   const m = , n = 85;
                  ^
Error    2: There must be a number to follow '='.
00001   var x,y,z,q,r;
00002
00003   procedure multiply;   (*note1 *)
00004   var a,b;
00005   begin
00006    a :=x;b :=y;z :=0;
00007    while b > 0 do
00008    begin
00009        if odd b then z := z + a;
00010        a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011    end
00012   end;
```

图 11: 测试 $error(2)$

```
00052  begin
00053    x := m; y := n; call multiply;
00054    x := 25; y := 3; call divide;
00055    x := 34; y := 36; call gcd;
00056  end
00057  .
There are 1 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 12: 测试 $error(2)$

- 测试 $error(3)$: "There must be an $'='$ to follow the identifier.",将第一行 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85;$ 改为 $const\ \ m,\ \ n\ \ =\ \ 85;$

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m, n = 85;
             ^
Error   3: There must be an '=' to follow the identifier.
00001  var x,y,z,q,r;
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006    a :=x;b :=y;z :=0;
00007    while b > 0 do
00008    begin
00009        if odd b then z := z + a;
00010        a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011    end
00012  end;
```

图 13: 测试 $error(3)$

```
00052  begin
00053    x := m; y := n; call multiply;
00054    x := 25; y := 3; call divide;
00055    x := 34; y := 36; call gcd;
00056  end
00057  .
There are 1 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 14: 测试 $error(3)$

- 测试 $error(4)$: "There must be an identifier to follow $'const'$, $'var'$, or $'procedure'$.", 将第一行 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85;$ 改为 $const;$

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const;
           ^
Error   4: There must be an identifier to follow 'const', 'var', or 'procedure'.
           ^
Error   7: Statement expected.
00001  var x,y,z,q,r;
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
00009       if odd b then z := z + a;
00010       a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011   end
00012  end;
```

图 15: 测试 $error(4)$

```
00052  begin
00053   x := m; y := n; call multiply;
00054   x := 25; y := 3; call divide;
00055   x := 34; y := 36; call gcd;
00056  end
00057  .
There are 2 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 16: 测试 $error(4)$

- 测试 $error(5)$: "There must be an identifier to follow 'const', 'var', or 'procedure'.", 将第一行 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85;$ 改为 $const\ \ m\ \ =\ \ 7,\ \ n\ \ =\ \ 85$

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85
00001  var x,y,z,q,r;
         ^
Error   5: Missing ',' or ';'.
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
00009       if odd b then z := z + a;
00010       a := 2 * a; b := b / 2;     (*note5as1d6a*)
00011   end
00012  end;
```

图 17: 测试 $error(5)$

```
00052  begin
00053   x := m; y := n; call multiply;
00054   x := 25; y := 3; call divide;
00055   x := 34; y := 36; call gcd;
00056  end
00057  .
There are 1 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 18: 测试 $error(5)$

2. **测试变量声明的语法分析：**由于其他报错与常量声明的语法分析报错差不多，因此这里不加赘述，只测试新加的 $error(26)$："Vardeclaration wrong, after var should be only identifier but not expression."。将第二行 $var\ x,y,z,q,r;$ 改为 $var\ x=a1=a2=1,y,z,q,r;$ 可以看见程序正确指出了所有不该有的赋值，并不影响之后的语法分析。

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/test.txt
00000  const m = 7, n = 85;
00001  var x = a1 = a2 = 1,y,z,q,r;
             ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
                 ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
                       ^
Error  26: Vardeclaration wrong, after var should be only identifier but not expression.
00002
00003  procedure multiply;   (*note1 *)
00004  var a,b;
00005  begin
00006   a :=x;b :=y;z :=0;
00007   while b > 0 do
00008   begin
```

图 19: 测试 $error(26)$

```
00052  begin
00053   x := m; y := n; call multiply;
00054   x := 25; y := 3; call divide;
00055   x := 34; y := 36; call gcd;
00056  end
00057  .
There are 3 error(s) in PL/0 program.

Compile End!
Program ended with exit code: 0
```

图 20: 测试 $error(26)$

3.2 对自己撰写的的 Bezout.txt 中源程序进行语法分析

源程序展示如下：

```
1 (*--UTF-8--*)
2 (*这是使用pl0语言编写的一个计算正整数a和b的Bezout等式的程序.*)
3 (*Created by 刘浩文 517021911065 on 2019/11/21.*)
4
5 const test = 1, ntest = 0;
6 var a, b, q, c, r, R, s, S, t, T;
7
8 procedure initial;  (*该过程将变量初始化.*)
```

```
begin
  q := 0; c := 0;
  r := a; R := b;
  s := 1; S := 0;
  t := 0; T := 1;
end;

procedure LOOP; (*该过程计算出a与b的最大公因数并保存在r中*)
begin
  while R <> 0 do
  begin
    c := S;
    S := -q * S + s;
    s := c;
    c := T;
    T := -q * T + t;
    t := c;
    q := r / R;
    c := R;
    R := -q * R + r;
    r := c;
  end
end;

begin     (*主函数, 输入a,b的数值得到其Bezout等式.*)
  a := 1847;
  b := 9832;
  call initial;
  q := r / R;
  c := R;
  R := -q * R + r;
  r := c;
  call LOOP;
end
.
```

使用完成的语法分析程序进行编译后输出结果如图 21、22。可见，语法分析程序能正确对自己撰写的 PL/0 源程序进行语法分析。

值得一提的是，这个 Bezout.txt 源程序是我在写词法分析程序时完成的，当直接用完成的语法分析程序对词法分析使用的 Bezout.xt 进行语法分析时，语法分析出现了大量报错。这是因为在写词法分析程序时并未考虑 PL/0 语言的语法，导致出现大量语法错误。在自己完成的语法分析程序的帮助下，我对 Bezout.txt 进行了 *Debug*，完成了没有语法错误的 PL/0 源程序。这也算是对自己的产品的一个使用实例吧。

```
Please input source file name:
    /Users/hongxing/Desktop/大三上学期/编译原理/大作业/语法分析大作业/Compile_Prin2/Compile_Prin2/Bezout.tx
00000  (*--UTF-8--*)
00001  (*这是使用pl0语言编写的一个计算正整数a和b的Bezout等式的程序.*)
00002  (*Created by 刘浩文 517021911065 on 2019/11/21.*)
00003
00004  const test = 1, ntest = 0;
00005  var a, b, q, c, r, R, s, S, t, T;
00006
00007  procedure initial;   (*该过程将变量初始化.*)
00008  begin
00009   q := 0; c := 0;
00010   r := a; R := b;
00011   s := 1; S := 0;
00012   t := 0; T := 1;
00013  end;
00014
00015  procedure LOOP;  (*该过程计算出a与b的最大公因数并保存在r中*)
00016  begin
00017   while R <> 0 do
00018   begin
00019       c := S;
00020       S := -q * S + s;
00021       s := c;
```

图 21: 语法分析 *Bezout.txt*

```
00022       c := T;
00023       T := -q * T + t;
00024       t := c;
00025       q := r / R;
00026       c := R;
00027       R := -q * R + r;
00028       r := c;
00029   end
00030  end;
00031
00032  begin            (*主函数，输入a,b的数值得到其Bezout等式.*)
00033   a := 1847;
00034   b := 9832;
00035   call initial;
00036   q := r / R;
00037   c := R;
00038   R := -q * R + r;
00039   r := c;
00040   call LOOP;
00041  end
00042  .\377

   success with no error!
Compile End!
Program ended with exit code: 0
```

图 22: 语法分析 *Bezout.txt*

4 实验中的问题与思考

- 没有 PL/0 语言的文法，仅凭提供的语法分析流程解释与不完整的代码不够完全理解语法分析程序的过程。提供的语法分析流程解释中只涉及到了非终结符，即主要 PL/0 语言的主要语法成分，以及这些非终结符的 $FIRST$ 集与 $FOLLOW$ 集；而提供的不完整的源程序中一是分析过程不完整，有些地方不明所以，二是由于递归下降分析法是多个函数相互递归调用，再加上使用单链表来存储预测单词符号，使对源码的阅读理解更为复杂。而递归下降分析法的过程如课堂和课本上介绍的那样，是完全基于文法的。所以为了正确完成大作业的任务，我在网上查找了相关资料，结合提供的语法分析流程解释与不完整的代码总结出了 PL/0 语言的文法，得以正确补全语法分析相关部分，如 1.2 节所述。

- 在完成本次大作业时，遇到的最大的问题还是自己一直使用的 $Xcode$ 集成开发环境在更新后突然就无法链接多个 $.c$ 源文件了，在网上查找的解决方法也都没有效果。无奈之下只能把 $pl0.c$ 和 $set.c$ 合并，才得以继续进行语法分析程序的设计。

5 感想与建议

5.1 感想

通过这次大作业，我综合了课堂所学的递归下降分析法理论知识，完成了一个语法分析程序，对语法分析形成了一个感性认识。之前学习理论知识的时候，只单单了解语法分析的原理与过程，对实际操作并没有成型的概念。且课堂上所学的递归下降分析法看起来非常简单，只是按照文法将一些函数相互递归调用而已。但这次大作业中发现，对函数对设计还是非常复杂的，要考虑多方面因素，还难以得出完美的解决方案。这次使用 C 语言写一个语法分析程序，让我对语法分析有了新的认识。这次的实践定将让我受益匪浅。

5.2 建议

建议提供的说明里不要仅仅是语法分析的流程介绍，最好还附上完整的 PL/0 语言的文法，因为递归下降分析法的过程是完全依赖于文法的，不知道正确的文法如同盲人摸象，好像自己在设计新的语言。