《编译原理》实验报告

实验二 语法分析

学院： 计算机学院

学号： 16121368

姓名： 刘洋

小组成员：刘锦 刘洋 陈杰 卿波

实验时间：2019年4月4日

## 实验目的

* 根据 PL/0 语言的文法规范，编写 PL/0 语言的词法分析程序。
* 通过设计调试词法分析程序，实现从源程序中分出各种单词的方法； 加深对课堂教学的理解；提高词法分析方法的实践能力。
* 掌握从源程序文件中读取有效字符的方法和产生源程序的内部表示 文件的法。
* 掌握词法分析的实现方法。
* 上机调试编出的词法分析程序。

## 实验内容：

输入 PL/0 语言程序，输出程序中各个单词符号（关键字、专用符号以及其它标记）。

## 实验要求：

* 词法分析器读入 PL/0 语言源程序（文本文件），识别结果也以文本 文件保存。
* 词法分析器的输出形式采用二元式序列，即：

(单词种类, 单词的值)

* 源程序中字符不区分大小写，即：“a1”和“A1”是同一个标识符。
* 准备至少 5 组测试用例，每组测试用例包括：输入源程序文件和输 出结果。
* 测试用例必须包含所有的基本字、运算符、界符、以及各种标识符 和常数。对不合法单词进行分类考虑测试用例，特别是对一些运算符要充分考虑各种组合。

## 算法描述：

本次实验小组用两种语言编程实现。

对于C++程序，算法描述如下：首先读入程序，按单词划分，其中，所有的运算符、界符、无符号整数、标识符、保留字都可看做一个单词，对于每一个单词，先判断是否为保留字和运算符以及界符，这些都是通过一个字典将符号串映射成编码，如果不是，再判断是否为标识符，最后再判断是否为数字，如果都不是，则把它看做null，最后输出最终结果。

对于python程序，算法描述如下：首先读入程序，变成小写字母字符串，按字符读取，利用一个临时存储的列表来存放连续字母或连续字母组成的单词或连续数字组成的数字，然后判断运算符，界符，这里把“:=”,“<=”, “>=”考虑到特殊性，分开处理，将所有单词都分好之后，存进一个列表中，再根据两个列表之间的映射，判断出具体的编码，当并不属于基本字、常数、运算符、界符等，剩下的单词作为标识符，之后输出最终结果。

## 实验代码：

C++程序如下：

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  class Analyzer  {  public:  string text;  string clas[29]= {"plus","minus","times","slash","oddsym","eql","neq","lss","leq","gtr","geq","lparen","rparen","comma",  "semicolon","period","becomes","beginsym","endsym","ifsym","thensym","whilesym","writesym","readsym",  "dosym","callsym","constsym","varsym","procsym"  };  string val[29]= {"+","-","\*","/","odd","=","#","<","<=",">",">=","(",")",",",";",".",":=","begin","end","if","then",  "while","write","read","do","call","const","var","procedure"  };  map<string,string> mp;  int idx;  Analyzer(string \_text);  virtual ~Analyzer();  string getNext();  bool isNone(char ch);  bool isNumber(string s);  bool isIdent(string s);  bool find(string s);  protected:  private:  };  int main()  {  string text="";  char ch;  freopen("in.txt","r",stdin);  freopen("out.txt","w",stdout);  while(scanf("%c",&ch)!=EOF)  {  text+=ch;  }  Analyzer analyzer(text);  vector<string> words;  vector<string> res;  string word=analyzer.getNext();  while(word!="")  {  words.push\_back(word);  if(analyzer.find(word))  {  res.push\_back(analyzer.mp[word]);  }  else if(analyzer.isIdent(word))  {  res.push\_back("ident");  }  else if(analyzer.isNumber(word))  {  res.push\_back("number");  }  else  {  res.push\_back("null");  }  word=analyzer.getNext();  }  for(int i=0; i<res.size(); i++)  {  cout<<"("<<setw(10)<<left<<res[i]<<","<<setw(10)<<right<<words[i]<<")"<<endl;  }  return 0;  }  Analyzer::Analyzer(string \_text)  {  //ctor  text = \_text;  idx=0;  for(int i=0; i<29; i++)  {  mp[val[i]]=clas[i];  }  }  Analyzer::~Analyzer()  {  //dtor  }  string Analyzer::getNext()  {  string tmp="";  while(idx<text.length()&&isNone(text[idx]))  {  idx++;  }  if(idx>=text.length()) return "";  tmp+=text[idx];  if(text[idx]>='a'&&text[idx]<'z')  {  idx++;  while((idx<text.length())&&(text[idx]>='a'&&text[idx]<='z'||text[idx]>='0'&&text[idx]<='9'))  {  tmp+=text[idx++];  }  }  else if(text[idx]>='0'&&text[idx]<='9')  {  idx++;  while(idx<text.length()&&text[idx]>='0'&&text[idx]<='9')  {  tmp+=text[idx++];  }  }  else if(text[idx]=='<')  {  idx++;  if(idx<text.length()&&text[idx]=='=')  {  tmp+=text[idx++];  }  }  else if(text[idx]=='>')  {  idx++;  if(idx<text.length()&&text[idx]=='=')  {  tmp+=text[idx++];  }  }  else if(text[idx]==':')  {  idx++;  if(idx<text.length()&&text[idx]=='=')  {  tmp+=text[idx++];  }  }  else  {  idx++;  }  return tmp;  }  bool Analyzer::isNone(char ch)  {  if(ch==' '||ch==10||ch==13||ch=='\t') return true;  return false;  }  bool Analyzer::isIdent(string s)  {  if(s==""||!(s[0]>='a'&&s[0]<='z')) return false;  for(int i=1; i<s.length(); i++)  {  if(!(s[i]>='a'&&s[i]<='z'||s[i]>='0'&&s[i]<='9')) return false;  }  return true;  }  bool Analyzer::isNumber(string s)  {  if(s=="") return false;  for(int i=0; i<s.length(); i++)  {  if(!(s[i]>='0'&&s[i]<='9')) return false;  }  return true;  }  bool Analyzer::find(string s)  {  return mp.find(s)!=mp.end();  } |

Python程序如下：

|  |
| --- |
| import re  f = open("test5.txt", "r")  # 把回车和两个空格都变成一个空格  string = f.read().lower().replace("\n", " ").replace(" ", " ")  print(string)  # 单个符号  d = ["+", "-", "\*", "/", ">", "<", "=", ":", "#", "(", ")", ",", ";", "."]  a = ["begin", "call", "const", "do", "end", "if", "odd", "procedure", "read", "then", "var", "while", "write"]  a1 = ["beginsym", "callsym", "constsym", "dosym", "endsym", "ifsym", "oddsym", "proceduresym", "readsym", "thensym",  "varsym", "whilesym", "writesy"]  b = ["+", "-", "\*", "/", "=", "#", "<", "<=", ">", ">=", ":=", "(", ")", ",", ";", "."]  b1 = ["plus", "minus", "times", "slash", "eql", "neq", "lss", "leq", "gtr", "geq", "becomes", "lapern", "rparen",  "comma", "semicolon", "period"]  allword = []  word = []  for i in range(len(string)):  if 'a' <= string[i] <= 'z':  word.append(string[i])  if 'a' <= string[i + 1] <= 'z' or '0' <= string[i + 1] <= '9':  continue  else:  word2 = "".join(word)  # print(word2)  allword.append(word2)  word.clear()  if '0' <= string[i] <= '9':  word.append(string[i])  if '0' <= string[i + 1] <= '9':  continue  else:  word2 = "".join(word)  print(word2)  allword.append(word2)  word.clear()  if string[i] == ':' and string[i + 1] == '=':  allword.append(":=")  continue  if string[i] == '<' and string[i + 1] == '=':  allword.append("<=")  continue  if string[i] == '>' and string[i + 1] == '=':  allword.append(">=")  continue  if string[i] == '=':  if string[i - 1] != '>' and string[i - 1] != '>' and string[i - 1] != ':':  allword.append(string[i])  for dd in d:  if string[i] == dd and string[i]!='=':  allword.append(string[i])  print(allword)  f2 = open('result2.txt','a')  for i in allword:  tag2 = 0  for j1 in range(len(a)):  if i == a[j1]:  tag2 = 1  break  # continue  for j2 in range(len(b)):  if i == b[j2]:  tag2 = 2  break  # continue  if tag2 == 1:  print("(", a1[j1], ",", i, ")")  f2.write("("+a1[j1]+ ","+i+ ")")  if tag2 == 2:  print("(", b1[j2], ",", i, ")")  f2.write("("+b1[j2]+ ","+ i+ ")")  if re.match("\d", i, re.I | re.S) != None:  tag2 = 3  print("(", "number", ",", i, ")")  f2.write("("+"number"+ ","+ i+ ")")  if tag2 == 0:  print("(", "ident", ",", i, ")")  f2.write("(" + "ident" + "," + i + ")")  f2.write("\n") |

## 实验结果：

1、测试样例1：

输入：

Const num=10;

Var a1,b2;

Begin

Read(A1);

b2:=a1+num;

const a=12;

write(A1,B2);

End.

输出：

( constsym , const )

( ident , num )

( eql , = )

( number , 10 )

( semicolon , ; )

( varsym , var )

( ident , a1 )

( comma , , )

( ident , b2 )

( semicolon , ; )

( beginsym , begin )

( readsym , read )

( lapern , ( )

( ident , a1 )

( rparen , ) )

( semicolon , ; )

( ident , b2 )

( becomes , := )

( ident , a1 )

( plus , + )

( ident , num )

( semicolon , ; )

( constsym , const )

( ident , a )

( eql , = )

( number , 12 )

( semicolon , ; )

( writesy , write )

( lapern , ( )

( ident , a1 )

( comma , , )

( ident , b2 )

( rparen , ) )

( semicolon , ; )

( endsym , end )

( period , . )

2、测试样例2：

输入：

var a,b,c,t;

procedure out;

begin

write(a);

write(b);

write(c);

end;

begin

read(a);

read(b);

read(c);

if a>b then

begin

t :=a;

a :=b;

b :=t;

end;

if a>c then

begin

t :=a;

a :=c;

c :=t;

end;

if b>c then

begin

t :=b;

b :=c;

c :=t;

end;

call out;

end.

输出：

( varsym , var )

( ident , a )

( comma , , )

( ident , b )

( comma , , )

( ident , c )

( comma , , )

( ident , t )

( semicolon , ; )

( proceduresym , procedure )

( ident , out )

( semicolon , ; )

( beginsym , begin )

( writesy , write )

( lapern , ( )

( ident , a )

( rparen , ) )

( semicolon , ; )

( writesy , write )

( lapern , ( )

( ident , b )

( rparen , ) )

( semicolon , ; )

( writesy , write )

( lapern , ( )

( ident , c )

( rparen , ) )

( semicolon , ; )

( endsym , end )

( semicolon , ; )

( beginsym , begin )

( readsym , read )

( lapern , ( )

( ident , a )

( rparen , ) )

( semicolon , ; )

( readsym , read )

( lapern , ( )

( ident , b )

( rparen , ) )

( semicolon , ; )

( readsym , read )

( lapern , ( )

( ident , c )

( rparen , ) )

( semicolon , ; )

( ifsym , if )

( ident , a )

( gtr , > )

( ident , b )

( thensym , then )

( beginsym , begin )

( ident , t )

( becomes , := )

( ident , a )

( semicolon , ; )

( ident , a )

( becomes , := )

( ident , b )

( semicolon , ; )

( ident , b )

( becomes , := )

( ident , t )

( semicolon , ; )

( endsym , end )

( semicolon , ; )

( ifsym , if )

( ident , a )

( gtr , > )

( ident , c )

( thensym , then )

( beginsym , begin )

( ident , t )

( becomes , := )

( ident , a )

( semicolon , ; )

( ident , a )

( becomes , := )

( ident , c )

( semicolon , ; )

( ident , c )

( becomes , := )

( ident , t )

( semicolon , ; )

( endsym , end )

( semicolon , ; )

( ifsym , if )

( ident , b )

( gtr , > )

( ident , c )

( thensym , then )

( beginsym , begin )

( ident , t )

( becomes , := )

( ident , b )

( semicolon , ; )

( ident , b )

( becomes , := )

( ident , c )

( semicolon , ; )

( ident , c )

( becomes , := )

( ident , t )

( semicolon , ; )

( endsym , end )

( semicolon , ; )

( callsym , call )

( ident , out )

( semicolon , ; )

( endsym , end )

( period , . )

3、测试样例3：

输入：

var x,y,a,b,c;

procedure fun;

begin

a:=1;

b:=2;

c:=3;

x:=a+b;

end;

begin

call fun;

end.

输出：

( varsym , var )

( ident , x )

( comma , , )

( ident , y )

( comma , , )

( ident , a )

( comma , , )

( ident , b )

( comma , , )

( ident , c )

( semicolon , ; )

( proceduresym , procedure )

( ident , fun )

( semicolon , ; )

( beginsym , begin )

( ident , a )

( becomes , := )

( number , 1 )

( semicolon , ; )

( ident , b )

( becomes , := )

( number , 2 )

( semicolon , ; )

( ident , c )

( becomes , := )

( number , 3 )

( semicolon , ; )

( ident , x )

( becomes , := )

( ident , a )

( plus , + )

( ident , b )

( semicolon , ; )

( endsym , end )

( semicolon , ; )

( beginsym , begin )

( callsym , call )

( ident , fun )

( semicolon , ; )

( endsym , end )

( period , . )

4、测试样例4：

输入：

Const a=10;

var b,c;

procedure p;

begin

c:=a+b

end;

begin

read(b);

while b#0 do

begin

call p;write(2\*c);read(b)

end

end.

输出：

( constsym , const )

( ident , a )

( eql , = )

( number , 10 )

( semicolon , ; )

( varsym , var )

( ident , b )

( comma , , )

( ident , c )

( semicolon , ; )

( proceduresym , procedure )

( ident , p )

( semicolon , ; )

( beginsym , begin )

( ident , c )

( becomes , := )

( ident , a )

( plus , + )

( ident , b )

( endsym , end )

( semicolon , ; )

( beginsym , begin )

( readsym , read )

( lapern , ( )

( ident , b )

( rparen , ) )

( semicolon , ; )

( whilesym , while )

( ident , b )

( neq , # )

( number , 0 )

( dosym , do )

( beginsym , begin )

( callsym , call )

( ident , p )

( semicolon , ; )

( writesy , write )

( lapern , ( )

( number , 2 )

( times , \* )

( ident , c )

( rparen , ) )

( semicolon , ; )

( readsym , read )

( lapern , ( )

( ident , b )

( rparen , ) )

( endsym , end )

( endsym , end )

( period , . )

5、测试样例5：

输入：

const a=10;

var b,c;

begin

read(b);

c:=a+b;

write(c)

end

输出：

( constsym , const )

( ident , a )

( eql , = )

( number , 10 )

( semicolon , ; )

( varsym , var )

( ident , b )

( comma , , )

( ident , c )

( semicolon , ; )

( beginsym , begin )

( readsym , read )

( lapern , ( )

( ident , b )

( rparen , ) )

( semicolon , ; )

( ident , c )

( becomes , := )

( ident , a )

( plus , + )

( ident , b )

( semicolon , ; )

( writesy , write )

( lapern , ( )

( ident , c )

( rparen , ) )

( endsym , end )

( period , . )

## 实验体会：

这次实验是关于词法分析器，帮助我们很好的认识了基本字、标识符、常数、运算符、界符。其实程序的思路还是很清楚的，难度也并不太大，小组内我们也试着用两种编程语言来共同实现这一功能。在编程的过程中，也遇到了一些问题，比如在处理特殊运算符“>=”，“:=”之类的，也是单独拿出来考虑。在C++程序里，也采用了vector类来帮助处理字符串，python中也使用了正则表达式re库用来判断常数、标识符，总的来说，这个实验还是很有意义的。