编译原理实验：语义分析及中间代码生成

原创

Todd222 于 2018-09-29 22:47:27 发布 21120 收藏 189

分类专栏： 编译原理实验 文章标签： 编译原理实验

本文链接：https://blog.csdn.net/Flamewaker/article/details/82903792

编译原理实验 专栏收录该内容

4 篇文章 8 订阅

订阅专栏

编译原理实验：语义分析及中间代码生成

1. 实验题目：语义分析及中间代码生成

实验目的

1. 通过上机实习，加深对语法制导翻译原理的理解，掌握将语法分析所识别的语法范畴变换为某种中间代码的语义翻译方法。
2. 掌握目前普遍采用的语义分析方法——语法制导翻译技术。
3. 给出PL/0文法规范，要求在语法分析程序中添加语义处理，对于语法正确的表达式，输出其中间代码；对于语法正确的算术表达式，输出其计算值。

实验内容

已给PL/0语言文法，在实验二或实验三的表达式语法分析程序里，添加语义处理部分，输出表达式的中间代码，用四元式序列表示。

实验要求

1. 语义分析对象重点考虑经过语法分析后已是正确的语法范畴，本实验重点是语义子程序。
2. 在实验二或实验三“语法分析器”的里面添加PL/0语言“表达式”部分的语义处理，输出表达式的中间代码，计算表达式的语义值。
3. 中间代码用四元式序列表示。

输入输出

1. PL/0算术表达式的语义计算：
   输入：
   PL/0算术表达式，例如: 2 + 3 * 5作为输入。
   输出：
   17
2. PL/0表达式的中间代码表示
   输入：
   PL/0表达式，例如: a *（b +c）
   输出：
   (+ b c t1)
   (* a t1 t2)

2. 设计思想

本次实验我采用的递归下降分析器的设计。

递归下降分析法的原理是利用函数之间的递归调用来模拟语法树自上而下的构建过程。从根节点出发，自顶向下为输入串中寻找一个最左匹配序列，建立一棵语法树。在不含左递归和每个非终结符的所有候选终结首字符集都两两不相交条件下，我们就可能构造出一个不带回溯的自顶向下的分析程序，这个分析程序是由一组递归过程（或函数）组成的，每个过程（或函数）对应文法的而一个非终结符。

语法：
<表达式> -> [+|-]<项>{<加法运算符> <项>}
<项> -><因子>{<乘法运算符> <因子>}
<因子> -> <标识符>|<无符号整数>|(<表达式>)
<加法运算符> -> +|-
<乘法运算符> -> *|/

计算FIRST集：
FIRST(<表达式>)={ +， -， (， <标识符>， <无符号整数> }
FIRST(<因子>)={ <标识符>， <无符号整数>， (}
FIRST(<项>)={ <标识符>， <无符号整数>， (}
FIRST(<加法运算符>)={ +， - }
FIRST(<乘法运算符>)={ *， / }

计算FOLLOW集：
FOLLOW(<表达式>)={) }
FOLLOW (<项>)={ +，- }
FOLLOW (<因子>)={ *，/ }
FOLLOW (<加法运算符>)={ <标识符>， <无符号整数>， (}
FOLLOW (<乘法运算符>)={ <标识符>， <无符号整数>， (}

可以改成如下拓广文法：

<表达式>-><项>|<表达式>+<项>|<表达式>-<项>

<项>-><因子>|<项>*<因子>|<项>/<因子>

<因子>->(<表达式>)|<标识符>|<无符号整数>

将语法消除左递归可得(用E代表<表达式>，T代表<项>，F代表<因子>)

E->TE’

E’->ε|+TE’|-TE’

T->FT’

T’->ε|*FT’|/FT’

F->ident|(E)|number

属性文法：是在上下文无关文法的基础上为每个文法符号（终结符或非终结符）配备若干个相关的“值”（称为属性）。

属性:代表与文法符号相关的信息，和变量一样，可以进行计算和传递。

综合属性

用于“自下而上”传递信息

在语法树中，一个结点的综合属性的值，由其子结点的属性值确定

S—属性文法：仅仅使用综合属性的属性文法

语义规则： 属性计算的过程即是语义处理的过程

对于文法的每一个产生式配备一组属性的计算规则，则称为语义规则。

(1)终结符只有综合属性，它由词法分析器提供

(2)非终结符既可以有综合属性也可以有继承属性，文法开始符号的所有继承属性作为属性计算前的初始值。

(3) 产生式右边符号的继承属性和产生式左边符号的综合属性都必须提供一个计算规则

(4) 产生式左边符号的继承属性和产生式右边符号的综合属性不由所给的产生式的属性计算规则进行计算，它们由其它产生式的属性规则计算

一遍扫描的处理方法： 与树遍历的属性计算方法不同，一遍扫描的处理方法是在语法分析的同时计算属性值，而不是语法分析构造语法树之后进行属性的计算，而且无需构造实际的语法树。

因为一遍扫描的处理方法与语法分析器的相互作用，它与下面两个因素密切相关：

1.所采用的语法分析方法

2.属性的计算次序

如下为递归下降分析器的设计实现思想

1. 对每个非终结符A构造一个函数过程，对A的每个继承属性设置一个形式参数，函数的返回值为A的综合属性（作为记录，或指向记录的一个指针，记录中有若干域，每个属性对应一个域）。
2. 非终结符A 对应的函数过程中，根据当前的输入符号决定哪个产生式候选。
3. 每个产生式对应的代码中，按照从左到右的次序，对于单词符号（终结符），非终结符和语义分析分别做以下的工作。
   （1）对于带有综合属性x的终结符X，把x的值存入为X.x设置的变量中。然后产生一个匹配X的调用，并继续读入一个输入符号。
   （2）对于每个非终结符号B，产生一个右边带有函数调用的赋值语句c=B(b1,b2,…，bk)
   （3）对于语义动作，把动作的代码抄进分析器中，用代表属性的变量来代替对应属性的每一次引用。

3.算法流程

1. 每一个非终结符对应于一个函数（子过程）；
2. 非终结符所对应的右侧产生式为函数体；
3. 每遇到一个终结符，则需要判断所输入字符是否与之匹配，若匹配则读取下一个，若不匹配，则进行出错处理。
   算法过程：

```
PROCEDURE <表达式>:
BEGIN
 IF SYM=’+’ OR SYM=’-’ THEN
 BEGIN
  ADVANCE; <项>;
  WHILE SYM=’+’ OR SYM=’-’ DO
  BEGIN
   ADVANCE; <项>;
  END
 END
 ELSE IF SYM=FIRST(<项>) THEN
 BEGIN
  <项>;
  WHILE SYM=’+’ OR SYM=’-’ DO
  BEGIN
   ADVANCE; <项>;
  END
 END
 ELSE ERROR
END

PROCEDURE <项>:
BEGIN
 IF SYM=’*’ OR SYM=’/’ THEN
 BEGIN
  ADVANCE; <因子>;
  WHILE SYM=’*’ OR SYM=’/’ DO
  BEGIN
   ADVANCE; <因子>;
  END
 END
 ELSE IF SYM=FIRST(<因子>) THEN
 BEGIN
  <因子>;
  WHILE SYM=’*’ OR SYM=’/’ DO
  BEGIN
          ADVANCE; <因子>;
        END
 END
 ELSE ERROR
END

PROCEDURE <因子>:
BEGIN
 IF SYM=’标识符’ OR SYM=<无符号整数> THEN
 BEGIN
ADVANCE;
 END
 ELSE IF SYM=’(’ THEN
 BEGIN
  <表达式>
  IF SYM=’)’ THEN
  BEGIN
   ADVANCE;
  END
  ELSE ERROR
 END
 ELSE ERROR
END
```

```
PROGRAM PAESER
BEGIN
 ADVANCE;
 <表达式>;
 IF SYM<>’#’ THEN ERROR
END
```

因此要在如上的基础上构造函数过程，函数中会包含计算属性。

4. 源程序

```
#include<bits/stdc++.h>
using namespace std;
ifstream infile("F:\\编译原理\\第四次实验\\result.txt");//文件流
ifstream infile2("F:\\编译原理\\第四次实验\\analysis.txt");//文件流
ofstream outfile("F:\\编译原理\\第四次实验\\result.txt");//文件输出流
map<string,string> word;//应用map数据结构形成一个string->string的对应
std::map<string,string>::iterator it;//用来遍历整个对应关系的迭代器
string str;//读入的字符串
string sym;//用来指示读入的符号
string sym2;//用来指示读入的符号
int count1=0,k=0,flag=0,conterr=0,lpnum=0;
string expressionAnalysis();//表达式分析，表达式的中间代码表示
string termAnalysis();//项分析，表达式的中间代码表示
string factorAnalysis();//因子分析，表达式的中间代码表示
int expressionAnalysis2();//表达式分析，算数表达式的语义计算
int termAnalysis2();//项分析，算数表达式的语义计算
int factorAnalysis2();//因子分析，算数表达式的语义计算
struct quad{//定义四元式
     string result;
     string arg1;
     string arg2;
     string op;
};
struct quad quad[50];
void map_init(){//对应关系进行初始化，如下只包括了表达式的相关符号
    word["+"]="plus";
    word["-"]="minus";
    word["*"]="times";
    word["/"]="slash";
    word["="]="eql";
    word["("]="lparen";
    word[")"]="rparen";
}
void lexanalysis(){//词法分析
    char ch;
    char a;
    string word1;//string变量识别单词
    string str;//string变量进行字符识别
    ostringstream buf;
    while(buf&&infile2.get(ch)) buf.put(ch);//将文件中的字符读出来
    str= buf.str();//将得到的字符储存到string类型变量中
    int csize=str.length();
    for(int i=0;i<csize;i++){//对整个字符串进行遍历
        while(str[i]==' '||str[i]=='\n') i++;//若最开始为空格或换行符，则将指针的位置往后移
        if(isalpha(str[i])){//对标识符和基本字进行识别,调用库函数isalpha()
            word1=str[i++];
            while(isalpha(str[i])||isdigit(str[i])){
                word1+=str[i++];
```

```
            }
            it=word.find(word1);
            if(it!=word.end()){//判断是不是基本字，若为基本字则进行输出
                outfile<<"("<<word[word1]<<","<<word1<<")"<<endl;
            }
            else{//否则直接输出
                outfile<<"(ident"<<","<<word1<<")"<<endl;
            }
            i--;
        }
        else if(isdigit(str[i])){//判断是不是常数，调用库函数isdigit()
            word1=str[i++];
            while(isdigit(str[i])){
                word1+=str[i++];
            }
             if(isalpha(str[i])){
                outfile<<"error"<<endl;
                break;
            }
            else{
                outfile<<"(number"<<","<<word1<<")"<<endl;
            }
            i--;
        }else{//对其他的基本字依次进行判断
            word1=str[i];
            it=word.find(word1);
            if(it!=word.end()){
                outfile<<"("<<word[word1]<<","<<word1<<")"<<endl;
            }else{
                outfile<<"error"<<endl;
                break;
            }
        }
    }
    infile2.close();
}
int advance(){//用来读入下一个符号
    int found1,found2;
    if(!getline(infile,str)){
        return 0;
    }
    found1=str.find(',',0);
    if(found1==-1){
        conterr++;
        cout<<"语法错误 识别字符错误"<<endl;
        return -1;
    }
    found2=str.length();
    sym=str.substr(1,found1-1);
    sym2=str.substr(found1+1,found2-found1-2);
    return 1;
}
string newtemp(){//产生新变量名t1,t2等
    char *p;
    char m[12];
    p=(char*)malloc(12);
    k++;
    itoa(k,m,10);
    strcpy(p+1,m);
```

```
    p[0]='t';
    string s;
    s=p;
    return s;
}
void emit(string op,string arg1,string arg2,string result){//产生四元式用于显示
    quad[count1].op=op;
    quad[count1].arg1=arg1;
    quad[count1].arg2=arg2;
    quad[count1].result=result;
    count1++;
    return;
}
string expressionAnalysis(){//表达式的递归下降分析程序
    string op,arg1,arg2,result;
    if(conterr){
        return NULL;
 }
 arg1=termAnalysis();//通过项分析得到第一个参数的值
 if(conterr){
        return NULL;
 }
 while((sym=="plus")||(sym=="minus")){
        op=sym2;
  flag=advance();
  if(conterr){
            return NULL;
  }
  if(flag==0){
            cout<<"语法错误 <加法运算符>后缺项"<<endl;
            conterr++;
   return NULL;
  }
  arg2=termAnalysis();//通过项分析得到第二个参数的值
  if(conterr){
            return NULL;
  }
  result=newtemp();//产生中间变量名，相当于对结果进行存储
  emit(op,arg1,arg2,result);//产生四元式，相当于进行加法或减法运算，得出结果
  arg1=result;//保存得到的结果
 }
 return arg1;//返回表达式最终得到的值
}
string termAnalysis(){//项的递归下降分析程序
    string op,arg1,arg2,result;
    arg1=factorAnalysis();//通过因子分析得到第一个参数的值
    if(conterr){
        return NULL;
    }
 while((sym=="times")||(sym=="slash")){
        op=sym2;
  flag=advance();
  if(conterr){
            return NULL;
  }
  if(flag==0){
   conterr++;
   cout<<"语法错误 <乘法运算符>后缺因子"<<endl;
   return NULL;
  }
```

```
  if(conterr){
            return NULL;
  }
  arg2=factorAnalysis();//通过因子分析得到第二个参数的值
  if(conterr){
            return NULL;
  }
  result=newtemp();//产生中间变量名，相当于对结果进行存储
  emit(op,arg1,arg2,result);//产生四元式，相当于进行乘法或除法运算，得出结果
  arg1=result;//保存得到的结果
 }
 return arg1;//返回项最终得到的值
}
string factorAnalysis(){
    string arg;
    if(sym=="ident"){//如果是标识符，最终返回标识符的符号
        arg=sym2;
        flag=advance();
        if(conterr){
            return NULL;
  }
  if(lpnum==0&&sym=="rparen"){
            conterr++;
   cout<<"语法错误 ')'不匹配"<<endl;
   return NULL;
        }
    }
    else if(sym=="number"){//如果是无符号整数，最终返回相应的整数
        arg=sym2;
        flag=advance();
        if(conterr){
            return NULL;
  }
  if(lpnum==0&&sym=="rparen"){
            conterr++;
   cout<<"语法错误 ')'不匹配"<<endl;
   return NULL;
        }
    }
    else if(sym=="lparen"){//如果是左括号，则要进行右括号匹配，并判断中间是不是表达式，并得出表达式的值进行返回
        lpnum++;
        flag=advance();
        if(conterr){
            return NULL;
  }
  if(flag==0){
   conterr++;
   cout<<"语法错误 '('后缺少表达式"<<endl;
   return NULL;
  }
        arg=expressionAnalysis();
        if(conterr){
            return NULL;
  }
        if(flag==0||sym!="rparen"){
   conterr++;
   cout<<"语法错误 表达式后面缺少')'"<<endl;
   return " ";
  }else{
```

```
      lpnum--;
            flag=advance();
            if(conterr){
                return NULL;
            }
            if(flag==0){
                return arg;
            }
  }
    }else{
  cout<<"语法错误 因子首部不为<标识符>|<无符号整数>|'('"<<endl;
  conterr++;
  return " ";
 }
 return arg;
}
int expressionAnalysis2(){//表达式的递归下降分析程序
    string op;
    int arg1,arg2,result;
    if(conterr){
        return 0;
 }
 arg1=termAnalysis2();//通过项分析得到第一个参数的值
 if(conterr){
        return 0;
 }
 while((sym=="plus")||(sym=="minus")){
        op=sym2;
  flag=advance();
  if(conterr){
            return 0;
  }
  if(flag==0){
            cout<<"语法错误 <加法运算符>后缺项"<<endl;
            conterr++;
   return 0;
  }
  arg2=termAnalysis2();//通过项分析得到第二个参数的值
  if(conterr){
            return 0;
  }
  if(op=="+"){//若是加法符号则进行加法运算，并对得到的结果进行保存
            result=arg1+arg2;
            arg1=result;
  }
        else{//若是减法符号则进行加法运算，并对得到的结果进行保存
            result=arg1-arg2;
            arg1=result;
        }
 }
 return arg1;//返回该表达式所代表的值
}
int termAnalysis2(){//项的递归下降分析程序
    string op;
    int arg1,arg2,result;
    arg1=factorAnalysis2();//通过因子分析得到第一个参数的值
    if(conterr){
        return 0;
    }
 while((sym=="times")||(sym=="slash")){
```

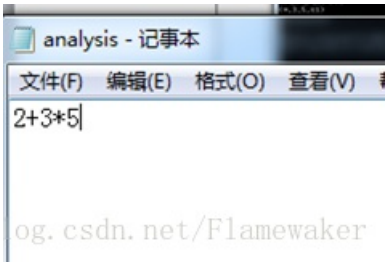
```
while((sym== "times" )||(sym== "slash" )){
        op=sym2;
  flag=advance();
  if(conterr){
            return 0;
  }
  if(flag==0){
   conterr++;
   cout<<"语法错误 <乘法运算符>后缺因子"<<endl;
   return 0;
  }
  if(conterr){
            return 0;
  }
  arg2=factorAnalysis2();//通过因子分析得到第二个参数的值
  if(conterr){
            return 0;
  }
  if(op=="*"){//若是乘法符号则进行加法运算，并对得到的结果进行保存
            result=arg1*arg2;
            arg1=result;
  }
        else{//若是除法符号则进行加法运算，并对得到的结果进行保存
            if(arg2==0){
                conterr++;
                cout<<"除数不能为0"<<endl;
                return 0;
            }
            result=arg1/arg2;
            arg1=result;
        }
 }
 return arg1;//返回该项所代表的值
}
int factorAnalysis2(){
    int arg;
    if(sym=="ident"){//算数表达式中不含有字母，否则无法进行运算
        cout<<"算术表达式中含有字母"<<endl;
  conterr++;
  return 0;
    }else if(sym=="number"){//若果是数字，则返回相应的值
        arg=atoi(sym2.c_str());
        flag=advance();
        if(conterr){
            return 0;
  }
  if(lpnum==0&&sym=="rparen"){
            conterr++;
   cout<<"语法错误 ')'不匹配"<<endl;
   return 0;
        }
    }
    else if(sym=="lparen"){//如果是左括号，则要进行右括号匹配，并判断中间是不是表达式，并得出表达式的值进行返回
        lpnum++;
        flag=advance();
        if(conterr){
            return 0;
  }
  if(flag==0){
   conterr++;
```

```
   cout<<"语法错误 '('后缺少表达式"<<endl;
   return 0;
  }
        arg=expressionAnalysis2();//返回表达式的值
        if(conterr){
            return 0;
  }
        if(flag==0||sym!="rparen"){
   conterr++;
   cout<<"语法错误 表达式后面缺少')'"<<endl;
   return 0;
  }else{
      lpnum--;
            flag=advance();
            if(conterr){
                return 0;
            }
            if(flag==0){
                return arg;
            }
  }
    }else{
  cout<<"语法错误 因子首部不为<标识符>|<无符号整数>|'('"<<endl;
  conterr++;
  return 0;
 }
 return arg;//返回该因子所代表的值
}
int main(){
    int i=0,num,result;
    //开始词法分析
    map_init();
    lexanalysis();
    //开始语法和语义分析
    cout<<"1.PL/0表达式的中间代码表示"<<endl;
    cout<<"2.PL/0算术表达式的语义计算"<<endl;
    cout<<"请输入类别号码:";
    cin>>num;
    flag=advance();
    if(num==1){//PL/0表达式的中间代码表示
        if(flag){
            expressionAnalysis();//开始进行表达式分析
        }
        if(flag!=-1&&!conterr){//若表达式正确则输出表达式的中间代码表示
            for(int i=0;i<count1;i++){
                cout<<'('<<quad[i].op<<','<<quad[i].arg1<<','<<quad[i].arg2<<','<<quad[i].result<<')'<<endl;;
            }
        }
    }else if(num==2){//PL/0算术表达式的语义计算
        if(flag){
            result=expressionAnalysis2();//开始进行表达式分析
        }
        if(flag!=-1&&!conterr){//若表达式正确则输出表达式的值
            cout<<result<<endl;
        }
    }else{
       cout<<"error!"<<endl;
       return 0;
    }
    infile.close();
```

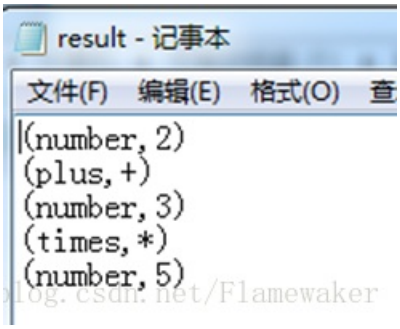
```
    infile.close();
    return 0;
}
```

5. 调试数据

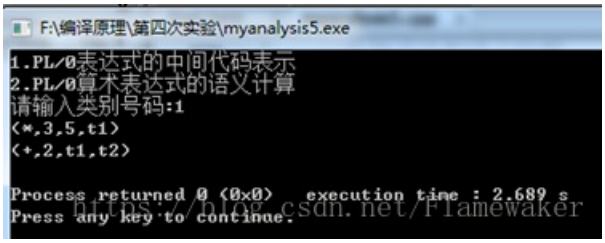
待输入的文件流：

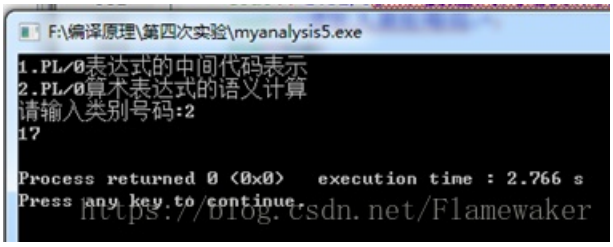


词法分析结果：



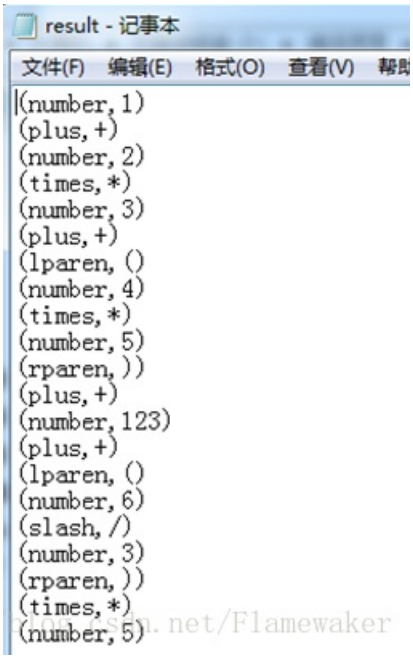
最终得到的结果：





待输入的文件流

词法分析结果：



最终得到的结果：