北京理工大學

文法分析实验报告

班　　级：07111505

姓　　名：徐宇恒

学　　号：1120151839

目录

一. 实验目的

语法分析是编译程序的核心部分。语法分析的任务是：按照语言的语法规则，对单词串形式的源程序进行语法检查，并识别出相应的语法成分。通过实验掌握语法分析的方法并理解语法分析在整个编译程序中的地位和重要性。

二. 实验内容

本次实验的内容，是要设计一个简单的 C 语言的语法分析程序模型，需要完成的功能为：依据 C 语言基本语法，处理词法分析器的输出，即属性字流形式的源程序，最终识别出无语法错误的语法成分。

构造语法分析器实现对经过词法分析生成的.xml 文件的分析，支持对于下述语句的处理：赋值语句、返回语句、一种分支语句和一种循环语句；同时支持对于运算符：+、-、*、/的识别和处理。语法分析器的输入、输出结果按照框架进行。

三. 实验环境

操作系统：Windows10 Pro

程序设计语言：C++

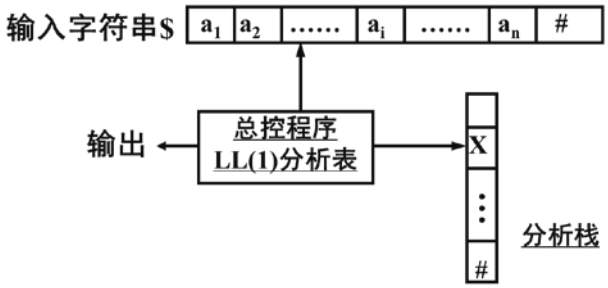
IDE：Visual Studio 2017

四. 实验方法

4.1. LL(1)分析法

LL(1)分析法是一种自上而下分析法，使用显式栈而不是递归调用来实现分析，判断读入的字符串是否属于该文法的某一句子。

LL(1)分析法的实现是由一个总控程序控制输入字符串在一张 LL(1)分析表和一个分析栈上运行而完成语法分析任务的。所以，一个 LL(1)分析器的逻辑结构如下图所示，由总控程序、LL(1)分析表和分析栈组成。



4.2. 实验基本思路

- 构建二维数组，即 LL(1)分析表
- 初始化用来表示分析栈的一维数组，依次将“#”和文法开始符号压入栈
- 读取经过词法分析得到的字符串作为输入串，以“#”结束
- 取出分析栈的栈顶字符 X，输入串的第一个字符 a
  - X 为终结符：
    - X=a=“#”，表示分析成功，停止分析过程；
    - X=a≠“#”，将 X 从分析栈顶退掉，指向输入字符串的指针下移一个字符；
    - X≠a，出错
  - X 为非终结字符，查分析表，对 M(X,a)：
    - 若 M(X,a)中为一个产生式规则，则将 X 从栈中弹出，并将此规则右部的符号序列推进栈；
    - 若 M(X,a)中为空白，表示出错。

五. 实验过程

9.1. 文法定义

文法定义是该实验的最基本步骤，文法定义的简单正确会为后续的编程奠定良好的基础。C 语言具有非常复杂的文法定义，有的文法的产生式很多，需要编译器根据读取的字符串进行移进和规约。我们这里仅仅对 C 语言中的部分文法进行代码实现。

使用文法如下

```
char input[17][MAX]={"E->TI(A){S}","T->i",
                     "I->m","I->a","I->b",
                     "A->GB","G->TI","B->UG",
                     "U->,","S->RP","R->r",
                     "P->IQ;","Q->OI","O->+",
                     "O->-","O->*","O->/"};
```

9.2. 构造 FIRST、FLLOW 集

根据构造的文法设计相应的 FIRST 集函数，需要对每个读入的非终结符进行处理，产生对应于每个非终结符的子集，相关代码如下：

```
void make_first ( )
 {
    memset ( used , 0 , sizeof ( used ) );
    for ( int i = 0 ; i < VN_set.size() ; i++ )
        dfs ( i );
#define DEBUG
#ifdef DEBUG
        puts ("***************FIRST集***********************");
        map<string,set<char> >::iterator it = first.begin();
        for ( ; it != first.end() ; it++ )
        {
            printf ( "FIRST(%s)={" , it->first.c_str() );
            set<char> & temp = it->second;
            set<char>::iterator it1 = temp.begin();
            bool flag = false;
            for ( ; it1 != temp.end() ; it1++ )
            {
                if ( flag ) printf ( "," );
                printf ( "%c" , *it1 );
                flag = true;
            }
            puts ("}");
        }
#endif
}
```

在构造 FIRST 集遇到产生式为空的情况需要产生 FOLLOW 集，主要使用 make_follow ( )函数对遇到空的形式进行推导，具体过程见源码文件。产生 FOLLOW 集的相应代码如下：

```
#ifdef DEBUG
    puts ("*****************FOLLOW集***********************" );
    map<string,set<char> >::iterator it = follow.begin();
    for ( ; it != follow.end() ; it++ )
    {
        printf ( "FOLLOW(%s)={" , it->first.c_str() );
        set<char> & temp = it->second;
        temp.insert('#');
        set<char>::iterator it1 = temp.begin();
        bool flag = false;
        for ( ; it1 != temp.end() ; it1++ )
        {
            if ( flag ) printf ( "," );
            printf ( "%c" , *it1 );
            flag = true;
        }
        puts ("}");
    }
#endif
```

9.3. 分析栈和生成树的构造

在构造生成树时，采用构造前序遍历的 N 叉树的形式产生，通过设计 Tree 类和结点 CTreeNode、CLeafNode 结构构造树，相关调用函数为 Tree：：系列函数。

9.4. Xml 文件生成

根据构造的 N 叉树，我们利用每个子节点与父节点之间的关系，采用前序遍历优先搜索，根据每个结点的属性构造相应的 xml 文件层次，相关代码如下：

六. 实验结果

通过对程序字符的输入，经过 LL（1）文法处理可以得到相关的 xml 文件：test.tree.xml。部分截图如下：

```
<PROGRAM>
  <Type>
    <keyword>int</keyword>
  </Type>
  <IDENT>
    <identifier>main</identifier>
  </IDENT>
  <separator>(</separator>
  <ARGS>
    <FRAGS>
      <Type>
        <keyword>int</keyword>
      </Type>
      <IDENT>
        <identifier>a</identifier>
      </IDENT>
    </FRAGS>
    <ALIST>
      <UNION>
        <separator>,</separator>
      </UNION>
      <FRAGS>
        <Type>
          <keyword>int</keyword>
        </Type>
        <IDENT>
          <identifier>b</identifier>
        </IDENT>
      </FRAGS>
    </ALIST>
  </ARGS>
  <separator>)</separator>
  <separator>{</separator>
  <STMTS>
          <keyword>return</keyword>
      <PLUS>
    <IDENT>
      <identifier>a</identifier>
    </IDENT>
    <CLOS>
      <OPER>
        <operarator>+</operarator>
      </OPER>
      <IDENT>
```

七. 心得体会

通过对语法分析器的编程设计，我可以明显感觉到语法分析器的设计实现明显难于词法分析器。在实验过程中遇到的主要问题是分析栈的构建以及如何构造一棵按照分析栈对字符的处理相对应的 N 叉树，最后还需要将 N 叉树的结点通过相应的结点关系与输出 xml 文件层次关系相对应。

完成了对语法分析器的设计，使我更加清楚地认识到编译程序如何按照文法规则识别并处理字符间的顺序关系，也让我认识到了一套好的文法定义将会为后续的语法语义处理带来很大的便利。这提醒我：一套好的理论可以为最后的设计提出最简单快捷的道路，同时，在设计过程中，我们也要不断找到方法去完善、实现这套理论。编写语法分析器让我意识到了编译程序的重要性与编写难度，也让我更加熟悉了语义处理文法的理论。