**《编译原理》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2018级物联网02班** | | **姓名** | **刘玉莽** |
| **实验题目** | 语法分析程序的设计与实现 | | | | |
| **实验时间** | **2021/5/16** | | **实验地点** | **DS1421** | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | |
| 一、实验目的  🡪了解编译器中语法分析环节的作用和意义  🡪掌握语法分析的方法和步骤  🡪熟悉自底向上、自顶向下等各类语法分析方法；掌握语法树的构造方法 | | | | | |
| 二、实验项目内容  一）实验内容  设计并实现实验一中的C语言子集的语法分析程序，要求实现以下功能：   1. 必做项： 2. 可以识别出用C语言子集中数据类型和语句集编写的源程序，并以语法分析树的形式输出分析结果。   （基本数据类型至少包含整数、布尔类型；至少包含赋值语句、变量申明语句、算数加/减法表达式、布尔表达式、循环语句、分支语句）   1. 检查源程序中存在的语法错误，并报告错误所在的位置。 2. 可选项：   需在给定C语言子集的基础上进行扩充,实现下列要求：   1. 能识别函数调用语句 2. 对源程序中出现的错误进行适当的恢复，使得语法分析可以继续进行，检查并报告源程序中存在的所有语法错误及错误位置   二）实验要求：   1. 自由选用程序设计语言（java，python，c/c++）作为实现语言，手工编写语法分析程序。 2. 提交实验报告及源代码。实验报告需严格遵循学校文档规范，内容包含对应文法、语法分析测试用例。   三）C语言子集  数据类型: **int**, 无符号整数, 取值范围0-9999  int a;  int a,b;  int a = 1;  算术运算符: **+**,**-**  a = b + 1;  a = b + c;  赋值运算符: **=**  a = 1;  关系运算符: **==** ,**>**,**<,<>,>=,<=**  a = (b==c);  a = (b>c);  a = (b<c);  逻辑运算符: **&&**, **||, !**  a = (b&&c);  a = (b||c);  a = (!b);  条件语句: **if**  if(a==b)  {  };  循环语句: **while**  while(a==b)  {  };  输入,输出: **get**,**put**  get(a);  put(a);  语句结束符: **;**  **加分项：**  条件语句 **if else**  if(a==b)  {  };  else  {  }; | | | | | |
| 三、实验过程或算法 | | | | | |
| 1. 首先，实验二是在实验一的基础上完成的，在实验一中，已经实现的词法分析器对应的c语言是一个包括宏定义，注释，函数调用，基本数据类型（char、void、int、float），基本运算（算数运算、赋值运算，关系运算、逻辑运算、位运算），for，while循环的子集。在实验二中，由于输出语法分析树是一个很复杂又很考验整体代码架构的工作，为了达到实验二的输出语法分析树的真正目的，在实验二中去掉了对科学计数法的支持。在实验二中所完成的主要工作如下：    1. 基本数据类型（char、void、int、float）。**还增加了对指针的支持**。      * 1. 基本运算（算数运算、赋值运算，关系运算、逻辑运算、位运算）。   2. for，while循环的子集。   3. 识别函数体的声明。   4. 能识别函数调用语句。   5. 对源程序中出现的错误进行适当的恢复，使得语法分析可以继续进行，检查并报告源程序中存在的所有语法错误及错误位置， 其中如果发现有语法错误，程序所采取的方法是忽略当前token，直到遇到合法的token。比如对if语句后面的条件的判断：      1. 采用**递归下降**的方法进行语法分析，因为相比于其他分析方法，递归下降的方法更适于手工编码，递归下降主要需要解决的一个问题就是消除左递归，这里以四则运算的文法分析为例：    1. 未消除左递归文法是：   <expr> ::= <expr> + <term>  | <expr> - <term>  | <term>  <term> ::= <term> \* <factor>  | <term> / <factor>  | <factor>  <factor> ::= ( <expr> )  | Num  可以看到存在直接左递归与间接左递归。   * 1. 消除左递归之后的文法为：   <expr> ::= <term> <expr\_tail>  <expr\_tail> ::= + <term> <expr\_tail>  | - <term> <expr\_tail>  | <empty>  <term> ::= <factor> <term\_tail>  <term\_tail> ::= \* <factor> <term\_tail>  | / <factor> <term\_tail>  | <empty>  <factor> ::= ( <expr> )  | Num   1. 语法分析树说明    1. 只需要运行一次”syntacticAnalysis.c”就可以得到语法分析树以及提示错误信息，结果：            1. 上述结果可以看到在语法分析树输出的同时，也会给出相应的语法错误，如未定义，但为了确保语法分析的继续进行，忽视了这样的错误，直到遇到相应正确的token。 2. 结果同样展示了给语法分析程序可以识别函数的调用， 3. **对于语法分析树本身，利用缩进来展示树的结构，树的中间节点是非终结符，叶子节点终结符。其中非终结符以类似标签的形式显示—— <非终结符>** 4. 这里由于语法分析本身的限制，会把main函数也当作一个声明函数对待    1. 以语句(statement)的语法树为例，他的语法分析树部分的实现方式：           5. 类似的语法分析树的构建还有很多，这里不一一列出，总的原则是按照文法，给出语法分析树。       6. 其中语法树结构体的实现如下：       7. 1. 其中construct()函数主要负责分析树的实例化。       8. 利用BFS打印语法分析树： 5. 错误处理    1. 对与语法分析中产生的错误，会给出相应的错误提示 | | | | | |
| 四、实验测试  测试说明：  用来测试的c程序如下：  struct s {      char name[20];      int id;  };  int fun(){      int la;      la = 0;      return la;  }  int main(){      // find the minimum val      int a, b, c;      int arr[3];      @  // undefined      if(a == b){          c = a + b;      }      else if(a < b){          c = b;      }      else{          c = fun();      }      while (a)      {          a--;      }      return 0;  } | | | | | |
| 五、实验总结   1. 调试结果与直接运行结果不同，这个确实是第一次遇到    1. 调试   IMG_256   * 1. 直接运行   IMG_257   * 1. 原因     while (!(feof(pFile)))  {      memset(src, 0, SRCMEM);      fgets(src, 499, pFile);      printf("src:%s", src);      compiler();    }  原因以及解决方案：文件读取不规范造成内存溢出，改为用fread()规范读取   1. 结构体指针申请内存失败   while (token != '}') {  treeNode \*TState = malloc(sizeof(TState));  construct(TState, "<statement>");  TN->child[0] = TState;  statement(TState);  }  由于上面的”treeNode \*TState = malloc(sizeof(TState));”而使得后面的对于TState指针的操作都会发生访问越界的情况，更糟糕的是由于编译器提示访问越界的发生实在下一个malloc()函数申请空间的时候，而本生这个代码量达到了1000多行，导致找这个错误找了很长时间，最终找到错误并且将错误语句修改为”treeNode \*TState = malloc(sizeof(treeNode));” | | | | | |