编译原理 L2 语法分析实验报告

181250086 刘佳月 181250086@smail.nju.edu.cn

● 实现功能：

读入词法单元流、判断输入程序是否匹配程序设计语言的语法规范，并在规范的情况下构建起输入程序的静态结构，否则输出错误。

● 实现步骤：

1. 编写 Flex 源代码的定义部分
    - 1.1 被“%{”和“%}”包含的内容引用库函数。
      引用”syntax.tab.h”（使用其中定义的 token）。
      定义多叉树结构体 Node（左子女右兄弟，成员变量还有 name, type，记录要输出的值的 union）。
      定义枚举类 NODE_TYPE（包括 LEX_INT, LEX_FLOAT, LEX_ID, LEX_TYPE, LEX_OTHER_TOKEN, SYNTACTIC_UNIT, SYNTACTIC_UNIT_EMPTY，方便根据 type 按需输出）。
      进行变量和函数声明。
      定义宏 YY_USER_ACTION（在每次执行前都维护词法单元的位置信息）。
    - 1.2 %option yylineno：使用 flex 内置维护的行号。
    - 1.3 定义别称：简化词法规则的书写。
2. Flex 源代码的规则部分：匹配到终结符后，action 部分创建一个节点（type 根据输出要求的不同设置节点的 type 和 union 的值），将终结符的属性设为该节点，并返回这个 token。
3. Flex 源代码的用户函数部分：
    - 3.1 createTokenNode 函数: 为匹配到的终结符创建节点, 将行号设置为 yylineno。
    - 3.2 getNum 函数：将八进制、十六进制的数字转换为十进制。
4. 编写 Bison 源代码的定义部分
    - 4.1 被“%{”和“%}”包含的内容引用库函数。
      引用”lex.yy.c”（使用其中的 yylex 函数）。
      进行变量和函数声明。
      引入外部变量。
    - 4.2 更改此法单元的属性值的类型为树节点。
    - 4.3 注明%locations，从而可以使用 yylloc，用于维护语法单元的位置。
    - 4.4 用%token<node>声明终结符，用%type<node>非终结符。
    - 4.5 规定开始符、优先级（写得越靠下，优先级越高）。需要加入 UMINUS 占位，因为取负的优先级高于乘除，减号低于乘除。通过 %nonassoc LOWER_THAN_ELSE 解决悬空 else 问题。
5. Bison 源代码的规则部分：包括具体的语法、相应的语义动作和错误恢复。将产生式右部各项作为子节点挂到产生式左部，传入父节点的行号，自底向上地构建出一颗语法树。
6. Bison 源代码用户函数部分：
    - 6.1 yyerror 函数：记录语法错误数量，指定错误输出内容。
    - 6.2 buildSyntaxTree 函数：创建父节点，设置好其行号，然后使用变长参数列表

va_list 依次添上子节点。

6.3 createSingleNode 函数：当产生式产生了空串时，调用该函数，仅创建一个节点，并设置节点的 type 为 SYNTACTIC_UNIT_EMPTY.

6.4 tree_search 函数：先序遍历语法树，当当前节点的 type 不为 SYNTACTIC_UNIT_EMPTY 时，才按照输出要求输出。

7. 编写 main.cpp 文件：

7.1 引入外部变量和外部函数。

7.2 接收文件名参数，调用 yyparse 进行语法分析，如果有错，bison 自动调用 yyerror 输出错误，如果没有语法错误，就调用 tree_search 输出语法树结构。

● 印象深刻的困难：

1. 多叉树的数据结构：如何创建节点、子节点如何挂到父节点、如何先序遍历。一开始把子节点和父节点连接起来后，没有把父节点赋值给词法单元的属性$$，出现段错误。写 buildSyntaxTree 和 createSingleNode 时忘了返回创建好的节点，也出现段错误。
2. 当产生式产生空串时，不能直接把产生式左部的词法单元的属性节点设为 NULL，否则无法继续规约，而应该创造一个没有子节点的节点（但其 type 特别设置，输出时不输出）。
3. 创建父节点时设置行号，既可以直接用第一个子节点（即该语法单元的第一个词素）的行号，也可以用 bison 内置的行号@$。
4. 提交后发现好几个用例输出的总行数少了，怀疑是树建得不对。最后发现是有一个产生式右部有 5 个节点，我建树时漏挂了第 5 个节点。改了以后总行数对了，但是多个用例都只 Match 了前面少数几行，甚至有几个只 match 了前 4 行，由于第 4 行一定是 specifier，猜测是 specifier 的那两个产生式错了，最后发现是节点的 name TYPE 拼成了 TPYE.
5. 提交后发现 hard test 8 9 10 过不了，原来是浮点数的精度不够，把 Node 的 union 中的记录小数的值的类型从 float 改为 double，就 AC 了。

● 精巧的设计：

1. 枚举类 NODE_TYPE
   可以轻松通过节点的 type 区分节点 SYNTACTIC_UNIT_EMPTY 不输出，其他类型按不同格式输出。
2. 节点 Node union 类型在需要时使用不同的值，当节点为 LEX_INT 时，用 int_constant，当节点为 LEX_FLOAT 时，用 float_constant，当节点为 LEX_ID 或 LEX_TYPE 时，用 str_constant.

```
5   typedef enum{
6       LEX_INT=0,
7       LEX_FLOAT=1,
8       LEX_ID=2,
9       LEX_TYPE=3,
10      LEX_OTHER_TOKEN=4,
11      SYNTACTIC_UNIT=5,
12      SYNTACTIC_UNIT_EMPTY
13  }NODE_TYPE;
14
15  struct Node{
16      struct Node* child;
17      struct Node* next_sib;
18      char name[32];
19      union{
20          int int_constant;
21          double float_constant;
22          char str_constant[32];
23      };
24      int lineno;
25      NODE_TYPE type;
26  };
```