# 编译原理第一次实验报告

141270022 刘少聪 141270037 汪值

第一次实验的任务是编写一个程序对使用 C--语言书写的源代码进行词法分析和语法分析，并打印分析结果。使用词法分析工具 GNU Flex 和语法分析工具 GNU Bison 来帮助完成实验，如果程序错误能够输出，如果不存在错误就输出语法树。

对实验任务进行分析，发现在熟悉 Flex 和 Bison 书写方法的基础上，整个实验可以分割成五个部分：词法正则表达式、语法规则合理性、语法树建立和输出、错误处理和恢复、八进制和十六进制数的识别:

1、 词法正则表达式

在书后的文法定义中已经给出了词法单元的内容，对于十进制的数，要在原来数字组合的基础上或一个单独的'0'浮点型和 ID 按照词法规则逐步匹配即可，同时对八进制数、十六进制数的正确和错误形式都要建立匹配的正则表达式，原因在错误处理和恢复里。在匹配了正确的正则表达式之后，需要根据词法单元的类型和内容创建叶子词法节点，之后返回一个语法单元名完成和语法单元的交互。

2、 语法规则合理性

对于语法规则，存在语法冲突（“移入\规约”冲突或者“移入\移入”冲突），解决的方法首先是确定结合性（左、右、不）和优先级，这样就能解决大部分的语法冲突问题，实验中的语法结合性和优先级按照书后的表中排列，之后对于提示出现的语法冲突，先使用 output 调试机制查看冲突位置，之后加入%proc 标记的语法单元解决冲突。

3、 语法树建立和输出

语法树的建立是本次实验的核心，由于 Bison 的语法分析器是自底向上进行分析的，也就是说语法树的建立是自底向上的，而输出是自顶向下的，所以不能边分析边输出，只能分配一定的空间存储语法树，之后再在最后统一输出。

语法树是一个多叉树，在包含行号、名称、内容三个公共部分的情况下，我们考虑了两种树的结构。第一种结构是在节点结构中建立两个节点的指针，分别是它的第一个子节点和右兄弟，在构造语法树的过程中，就可以根据语法规则依据第一个子节点建立根节点，并将顺序的各个语法单元通过右兄弟连接起来。第二种结构是在节点结构中建立一个子节点指针数组，在构造语法树的过程中，首先初始化根节点，之后根据一个语法规则，将所有的子节点按照顺序添加到根节点上。两种方法均能够成功建立语法树，提交的实验代码使用的是第一种语法树的结构。

语法树的输出，就是从根节点开始，先序遍历语法树，根据每个节点的层数确定缩进位数，之后输出节点的名称、行号或者内容，使用的是递归的遍历方法。

4、 错误处理和恢复

对于检测中出现的错误，错误类型分为两种 :词法错误 (Type A) 和语法错误 (Type B)。词法分析对词法错误报错，语法分析对语法错误报错并且恢复，不影响整个语法树的构造，设置了一个全局变量存储总的错误个数，当存在错误时，不输出语法树。

对于词法错误，最常见的是出现无法识别的字符，处理方法是对所有不能识别的符号，用正则表达式“.”来识别，对这样的词法单元输出词法识别错误。除去非法字符，比较特别的是对于不合法的正整数的判断，由于需要识别十进制、八进制、十六进制，所以整数有多种格式，如果仅仅识别正确的词法单元，将会出现问题，比如“i=0909”这样的一个字符串，其中存在非法的八进制字符，如果直接识别将被识别为一个 int 型的“0”和一个 int 型的“909”，这就将词法错误误解析成了语法错误，所以需要对非法的八进制数和十六进制数

匹配单独的正则表达式。

对于语法错误，实验指导书上给出了 Bison 的处理机制，就是回退并寻找符合要求的包括 error 的语法规则，之后继续进行分析。实际上我们想要定位出错的语法单元，需要在尽量底层的产生式中添加包含 error 的语法规则，在实验的实现过程中，我们首先根据自己的理解在出现')'、']'、'}'、';'这些符号的语法中加入 error 语句进行判断，在消除语法冲突（“移入\规约”冲突或者“移入\移入”冲突）之后使用现实中出现的各种错误进行检验，最后确定了本次实验中的语法中可以检测出的错误以及错误类型。当 Bison 检查到了语法错误之后，将自动调用函数 yyerror，我们重写了这个函数保证没有输出，并重新构造了自己的错误输出函数来输出错误信息。

5、八进制和十六进制数的识别

我们是 11 组，所以在基本任务的基础上选做 1.1 的内容，就是识别八进制和十六进制数。首先八进制和十六进制数的正则表达式以及错误表达式要在词法分析中说明，上面已经解释了，而且实验要求将识别到的数字转化为十进制的数字存放在语法树中，这就需要对数字的进制进行转换，出于方便，在语法树的内容中存放的是字符串，所以最后要转化为字符串输出。调用 strtol(yytext,&end,16)这个内置函数将 16 进制字符串转化为 int 型的十进制数字，之后将这个数输出到 yytext 中，之后采用和十进制相同的方法创建叶子词法节点，八进制方法同十六进制。

通过完成上面的五个部分，实验一所要求的内容已经全部实现，对于真实编译器的情况，我们还没有考虑。实验中还遇到了一些小问题，在以后的代码书写中可以引以为鉴：

1、变长参数的使用

在树的构造过程中，开始的时候没有使用变长参数，导致书写的过程中生成树的函数或者语法规则部分总是过程而且冗余，发现小的问题要修改很多重复的语句，后来使用了变长参数 va_list，成功将重复的部分缩短，使代码更加简洁易懂。

2、char*指针

在语法树的每个节点的构造过程中，在最开始，我们使用的方法先声明了一个字符数组，之后传递参数进入构造节点的函数中，这样导致的结果就是每个节点的名称只是引用了这个字符数组，并没有将真实的字符串传递进去，在后来直接将字符串输入函数中，虽然会出现 string 和 char*的转换警告，但是能够保证字符串内容被传递进去。

3、Node*指针

在语法树构造的开始，我们将每个语法单元定义为一个 Node*类型的 node 实例，这样的结果就是要么系统没有给实例分配内存报错，要么系统已经创建了这样一个 node，我们对它的操作得不到正确的响应，结果就是语法树的各层之间不能连续连接，无法生成语法树。后来将语法节点定义为一个 Node*的类型，通过指针的传递成功实现了语法树构造。