编译原理 实验 1

161220013 陈金池

jcchen.gm@gmail.com

1、 简介

实验 1 使用 Flex、Bison 工具，实现了一个 c--语言的词法、语法分析的程序。除了基本语法外，该程序还实现了识别指数形式浮点数的功能。

2、 运行方法

在/Lab/Code 文件夹中执行以下指令：

make clean && make

此时已经生成 parser 可执行文件，可以选择分析测试用例

make test

或分析指定文件

./parser [path-to-cmm-file]

3、 实现细节

(1) 词法分析

该部分主要需要向 flex 提供所有终结符号的正则表达式，大部分内容在 lexical.l 文件中。

首先，为了方便书写，定义了一些常用的正则表达式：

```
positive-digit [1-9]
digit [0-9]
letter [a-zA-Z]
letterOrUs [a-zA-Z_]
delim [ \t\n\r]
```

各个表达式的名称表示其含义。

其次，除了已经提供的终结符表达式外，还需要完成 int、id 和 float（包括指数形式）的定义。其中前两个比较简单，此处不再赘述。float 的定义可以划分为**普通形式、指数形式**两部分，分别写出表达式后用“|”结合起来即可。对于普通形式，由于小数点前后均要有数字（而这些数字的形式没有其他要求），因此可以比较简洁地写出来：

```
{digit}+\.{digit}+
```

而对于指数形式，基数部分要求小数点前、后至少有一边有数字，所以需要考虑两种情况：

```
({digit}*\.{digit}+)|({digit}+\.{digit}*)
```

指数部分则包含一个可选的正负号、和一串数字：

```
(\+|-)?)({digit}+)
```

基数、指数部分通过一个指数符号（e|E）连接起来，即成为一个表示指数形式浮点数的整体。

此外，为了简化代码，定义了一个宏定义用于将信息保存至 yylval 中：

```
#define TOKEN_ASSIGN_YYLVAL(token_type) \
yylval=create_token_node(token_type,yylineno);
```

此处的 yylval 已经被定义为树节点类型，而 create_token_node()函数位于 TreeNode.c 文件中，用于生成终结符节点。

(2) 语法分析

该部分主要需要向 bison 提供文法，大部分内容在 syntax.y 文件中。

首先，在 syntax.y 文件中定义了各个 token，并指明其优先级和结合性。这些 token 在文法分析模块返回，传入语法分析模块进行规约。一个**特例**是，在文法分析中负号和减号都识别为减号，但两者的优先级并不相同，为了解决这个问题，在 token 定义时新增了 NEG 符号（根据优先级放在特定位置），然后在“-”表示负号的表达式中手动指明其优先级，以区分减号：

```
| LP Exp RP {$$ = create_nonte
| MINUS Exp %prec NEG {$$ = cr
| NOT Exp {$$ = create_nonterm
| ID LP Args RP {$$ = create_n
```

其次，为了构造语法树，将 YYSTYPE 定义为数据结构 struct TreeNode*，该指针指向的结构体表示一个非终结符或终结符（文法分析时 yylval 的值即为一个指向终结符的指针），包括其类型、第一行行号、子节点等内容。因此，每个语法的语义动作其实是“生成新节点，并将其子节点保存到结构体中”的过程。

此外，为了让程序在遇到错误时能够恢复并继续分析，在原有语法的基础上还添加了**错误恢复**部分。典型的几个错误恢复如下：

```
ExtDef :
    | error SEMI
    | Specifier error
    | Specifier ExtDecList error
```

这些内容表示在全局变量定义时“进行初始化赋值”、“没有分号”等错误。

```
Stmt : error SEMI
```

这一句表示“语句末尾没有分号”的错误。更多错误恢复语句此处不赘述。

4、其他

(1)TreeNode 结构

为了构造语法书，自定义了该数据结构（TreeNode.h 和 TreeNode.c），并提供一系列方便构造树的函数。

节点有一个 union 类型的值 value，用于保存整型（如 int 终结符）、浮点型（如 float 终结符）或字符串指针（如 id 和 relop 终结符）；有 is_token 和 type 变量，用来确定其具体类型；有一个子节点数组 children；还有一个保存其出现行号的 first_line 变量。

TreeNode.h 中还定义了一系列用于生成节点的函数，如 create_nonterminal_node 函数用于生成非终结符节点。在该函数中，因为子节点的个数不固定，使用了**可变参数**简化使用：

```
TreeNode *create_nonterminal_node
(int n_type, int first_line, int num_of_children, ...)
```

(2)语法树打印

语法树打印通过递归实现。定义如下：

void print_node(TreeNode *node, int depth);

具体而言，初始从 Program 节点出发，对某一个节点首先打印其自身的信息（通过 get_literal()方法构造该字符串），然后在进一步缩进（depth++）的位置打印它的所有子节点。其中，在打印前还需要判断该节点是否为空串规约而来，若是则直接返回（既不打印其自身，也不打印子节点）。